CAN-CBX-THERMO

4 Thermocouple Interfaces

Manual

to Product C.3034.02

esd electronics gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
http://www.esd.eu
Phone: +49 (0) 511 3 72 98-0 • Fax: +49 (0) 511 3 72 98-68

<u>N O T E</u>

esd electronics gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

Phone:	+49-511-372 98-0
Fax:	+49-511-372 98-68
E-mail:	info@esd.eu
Internet:	www.esd.eu

This manual contains important information and instructions on safe and efficient handling of the module. Carefully read this manual before commencing any work and follow the instructions.
The manual is a product component, please retain it for future use.

Trademark Notices

Document-File:	I:\Texte\Doku\MANUALS\CAN\CBX\THERMO\Englisch\CAN-CBX-Thermo_Manual_en_12.wpd
Date of print:	2019-02-07

PCB version:	CAN-CBX-THERMO Rev. 1.0
Firmware version:	V 2.01

Changes in the chapters

The changes in the document listed below affect changes in the hardware and firmware as well as changes in the description of facts only.

Revision	Chapter	Changes versus previous version
1.0	-	First English version
1.1	-	Safety Information revised, Typographical Conventions inserted
	3.	Technical data revised
	4.2, 4.3	Chapter revised
	5.4	Conductor cross section for 24V-connector corrected
	6., 7.	Chapters revised
	8.	Chapter restructured, Chapter “Communication Profile Area” inserted
	9.	References revised
	11.	EU-Conformity updated
1.2	-	Safety Instructions revised and inserted, Classification inserted
	2.	Chapter revised
	3.1	Description of housing added
	8.11.10.2	New chapter: “Disable Cold Junction Compensation (2425_h)”
	12	Order information of accessories supplemented

Technical details are subject to change without further notice.

Classification of Warning Messages and Safety Instructions

This manual contains noticeable descriptions, warning messages and safety instructions, which you must follow to avoid personal injuries or death and property damage.

This is the safety alert symbol.
It is used to alert you to potential personal injury hazards. Obey all safety messages and instructions that follow this symbol to avoid possible injury or death.

DANGER, WARNING, CAUTION

Depending on the hazard level the signal words DANGER, WARNING or CAUTION are used to highlight safety instructions and warning messages. These messages may also include a warning relating to property damage.

DANGER
Danger statements indicate a hazardous situation which, if not avoided, will result in death or serious injury.

WARNING
Warning statements indicate a hazardous situation that, if not avoided, could result in death or serious injury.

CAUTION
Caution statements indicate a hazardous situation that, if not avoided, could result in minor or moderate injury.

NOTICE

Notice statements are used to notify people on hazards that could result in things other than personal injury, like property damage.

NOTICE
This NOTICE statement indicates that the device contains components sensitive to electrostatic discharge.

NOTICE
This NOTICE statement contains the general mandatory sign and gives information that must be heeded and complied with for a safe use.

INFORMATION

INFORMATION
Notes to point out something important or useful.

Safety Instructions

- When working with CAN-CBX modules follow the instructions below and read the manual carefully to protect yourself and the CAN-CBX module from damage.
- Do not use damaged or defective cables to connect the CAN-CBX module and follow the CAN wiring hints in chapter: ”Correct Wiring of Electrically Isolated CAN Networks” .
- In case of damages to the device, which might affect safety, appropriate and immediate measures must be taken, that exclude an endangerment of persons and domestic animals and property.
- Current circuits which are connected to the device have to be sufficiently protected against hazardous voltage (SELV according to EN 60950-1).
- The CAN-CBX module may only be driven by power supply current circuits, that are contact protected. A power supply, that provides a safety extra-low voltage (SELV) according to EN 60950-1, complies with this conditions.

- Do not open the housing of the CAN-CBX module.
- The CAN-CBX module has to be securely installed before commissioning.
- The permitted operating position is specified as shown (Fig. 9). Other operating positions are not allowed.
- Never let liquids get inside the CAN-CBX module. Otherwise, electric shocks or short circuits may result.
- Protect the CAN-CBX module from dust, moisture and steam.
- Protect the CAN-CBX module from shocks and vibrations.
- The CAN-CBX module may become warm during normal use. Always allow adequate ventilation around the CAN-CBX module and use care when handling.
- Do not operate the CAN-CBX module adjacent to heat sources and do not expose it to unnecessary thermal radiation. Ensure an ambient temperature as specified in the technical data.

Qualified Personnel
This documentation is directed exclusively towards qualified personnel in control and automation engineering.
The installation and commissioning of the product may only be carried out by qualified personnel, which is authorized to put devices, systems and electric circuits into operation according to the applicable national standards of safety engineering.

Conformity
The CAN-CBX module is an industrial product and meets the demands of the EU regulations and EMC standards printed in the conformity declaration at the end of this manual.

Warning: In a residential, commercial or light industrial environment the CBX-module may cause radio interferences in which case the user may be required to take adequate measures.

Intended Use
The intended use of the CAN-CBX module is the operation as a CANopen-Slave with thermocouples. The esd guarantee does not cover damages which result from improper use, usage not in accordance with regulations or disregard of safety instructions and warnings.

- The CAN-CBX module is intended for indoor installation only.
- The operation of the CAN-CBX module in hazardous areas, or areas exposed to potentially explosive materials is not permitted.
- The operation of the CAN-CBX module for medical purposes is prohibited.

Service Note
The CAN-CBX module does not contain any parts that require maintenance by the user. The CAN-CBX module does not require any manual configuration of the hardware. Unauthorized intervention in the device voids warranty claims.

Disposal
Devices which have become defective in the long run have to be disposed in an appropriate way or have to be returned to the manufacturer for proper disposal. Please, make a contribution to environmental protection.

Typographical Conventions

Throughout this manual the following typographical conventions are used to distinguish technical terms.

Convention	**Example**
File and path names	`/dev/null or <stdio.h>`
Function names	***open()***
Programming constants	`NULL`
Programming data types	`uint32_t`
Variable names	*Count*

Number Representation
All numbers in this document are base 10 unless designated otherwise. For hexadecimal numbers $_h$ is appended . For example, 42 is represented as $2A_h$ in hexadecimal format.

Contents

1. Overview

1.1 Description of the Module

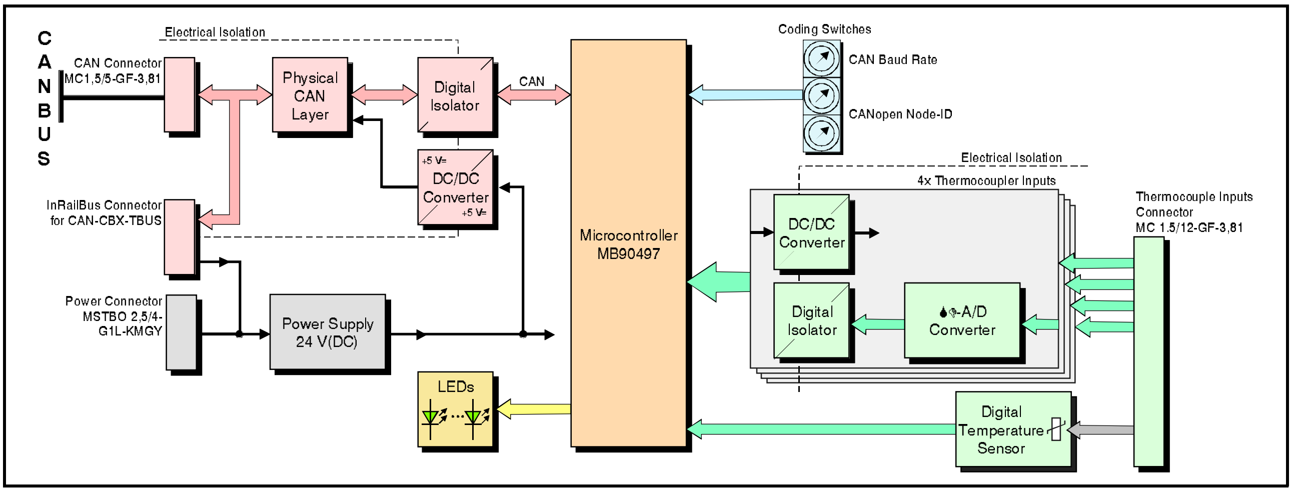


Fig. 1: Block circuit diagram of the CAN-CBX-THERMO module

The CAN-CBX-THERMO module is a CAN-CBX module with four High Resolution Thermocouple Interfaces.
The CAN-CBX-THERMO is equipped with four independent sigma-delta A/D converters for the evaluation of thermocouples. It features support of J, K, B, E, N, R, S and T thermocouples.
Depending on the selected sample rate and the external wiring a resolution of at least 1 µV can be achieved.
For cold junction compensation the temperature of the sensor clamp is measured by a digital temperature sensor.

The conversion of the four thermocouple inputs is realized by four independent ΣΔ-converters. Linearisation according to NIST is achieved by the on board microcontroller.

The CAN interface is designed according to ISO11898-2 high-speed layer with electrical isolation and supports bit rates up to 1 Mbit/s. The CANopen-node number and the CAN-bit rate can be easily set via coding switches.

The CAN-CBX-THERMO features the possibility to connect the power supply and the CAN bus signals via the InRailBus connector (TBUS-connector) integrated in the mounting rail. Individual modules can then be removed without interrupting the bus signals.

The module comes with CANopen® firmware according to CiA® 301 and supports the CiA 404 profile for measuring devices.

2. Quick Start

Step	Action	See page
	Read the safety instructions at the beginning of this document carefully, before you start with the hardware installation!	5
	DANGER Hazardous Voltage - Risk of electric shock. All current circuits which are connected to the device have to be sufficiently protected against hazardous voltage (SELV according to EN 60950-1).	
	NOTICE **Please note the chapters "Installation and Wiring of the Module" and "Correct Wiring of Electrically Isolated CAN Networks"!**	25, 35
1	Mount the CAN-CBX-module and connect the interfaces (power supply voltage, CAN, thermocouple inputs).	16
2	Please note that the CAN bus has to be terminated at both ends! esd offers special T-connectors and termination connectors. Additionally the CAN_GND signal has to be connected to earth at exactly one point in the CAN network. For details please read chapter "Correct Wiring of Electrically Isolated CAN Networks". A CAN node with electrical connection to earth potential acts as an earth potential.	35
3	Set the baud rate (only if it differs from the default setting) The default baud rate is 1 MBit/s. The baud rate can be set via the coding switch BAUD, as described in chapter: "Setting the Node-ID via Coding Switch".	24
4	Set the module number (node-ID). The node ID can be set via the coding switches LOW and HIGH. It may be set to values between 1 and 127 ($01\text{-}7F_h$).	23
5	Apply the 24 V power supply voltage.	-
6	Write "-1" ($FFFF\ FFFF_h$) in object 9133_h (sub-index 1...4) (With every new A/D-conversion a TxPDO with the measured value is sent.)	-

7	Send NMT-Start command to module The PDOs are sent on 0180_h + Module number 0280_h + Module number 0380_h + Module number 0480_h + Module number The Process-Value (PV) is sent in the data bytes 0...3. The default unit of the PV is µV. E.g.: 1234567 equates 1234,567 mV	-

3. Technical Data

3.1 General technical Data

Power supply voltage	nominal voltage: 24 V/DC input voltage range: 24 V ±20% current consumption (24 V, 20 ˚C): typical: 90 mA
Connectors	24V (4-pin line connector with spring-cage connection, X400) - 24V-power supply voltage InRailBus (5-pin CAN-CBX-TBUS connector, Phoenix Contact, X101) - CAN interface and power supply voltage via InRailBus S1...S4 (12-pin line connector with screw connection, X500) - 4 high resolution thermocouple interfaces CAN (5-pin line connector with spring-cage connection, X400) - CAN interface Only for test and programming purposes: X200 (6-pin connector) - the connector is placed inside the case
Temperature range	-20 ˚C ... +60 ˚C ambient temperature
Humidity	max. 90%, non-condensing
Protection class	IP20
Pollution degree	maximum permissible according to DIN EN 61131-2: Pollution Degree 2
Housing	Phoenix “MEMAX” plastic housing for carrier rail mounting NS35/7,5 DIN EN 60715
Dimensions	width: 22.5 mm, height: 99 mm, depth: 114,5 mm (including mounting rail fitting and connector projection, without mating plug)
Weight	145 g

Table 1: General technical data

3.2 Microcontroller

Microcontroller	16 bit µC MB90F497
RAM	2 Kbyte integrated
Flash	64 Kbyte integrated
EEPROM	minimum 256 byte

Table 2: Microcontroller

3.3 CAN Interface

Number	1
Connection	5-pin line connector with spring-cage connection or via InRailBus-connector (CAN-CBX-TBUS)
CAN Controller	MB90F497, ISO11898-1 (CAN 2.0)
Electrical isolation of CAN interfaces against other units	Isolation voltage U: 500 V (= withstand-impulse voltage according to DIN EN 60664-1)
Physical layer CAN	physical layer according to ISO 11898-2, transfer rate programmable from 10 Kbit/s up to 1 Mbit/s
Bus termination	has to be set externally if required

Table 3: Data of the CAN interface

3.4 Thermocouple Interfaces

Number	4 independent ΣΔ A/D-converter channels
Cold junction compensation	via internal digital temperature sensor, temperature measured at the sensor clamp
Sensor types	- J, K, B, E, N, R, S or T thermocouples with cold junction compensation - voltage measurement
Accuracy	sensor dependent (< 0.1˚C)
Input impedance	> 1 MΩ
Conversion rate	2.5 Hz ... 1000 Hz
Resolution	<1 µV at 25 Hz conversion rate ≈< 0.1˚C
Maximum sensor voltage	± 1.024 V
Electrical isolation	electrical isolation of thermocouple inputs against each other and against power supply
Connector	12-pin line connector with screw connection

Table 4: Data of analog inputs

3.5 Software Support

The firmware of the module comes with CANopen firmware according to CiA 301 [1] and supports the CiA 404 [4] profile for measuring devices.

The CAN-CBX-THERMO EDS-file can be downloaded from the esd website www.esd.eu.

4. Hardware Installation

4.1 Connecting Diagram

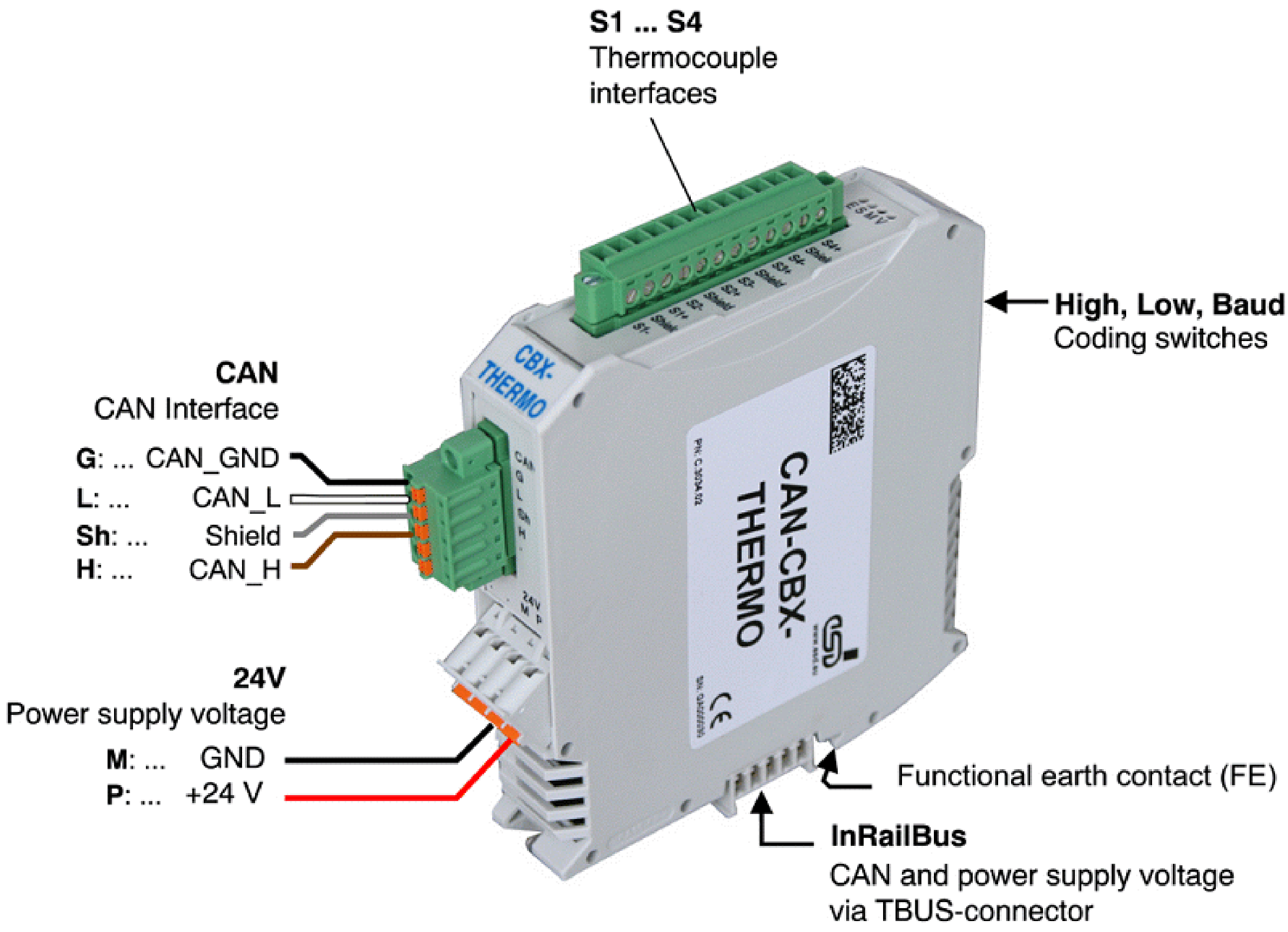


Fig. 2: Connections of the CAN-CBX-THERMO module

NOTICE
Read chapter "Quick Start" on page 11, before you start with the installation of the hardware!

INFORMATION
Refer to page 34 for information on conductor connection and conductor cross section. The connector pin assignments can be found on page 29 and the following.

4.1.1 Connection of Thermocouple (Example S1)

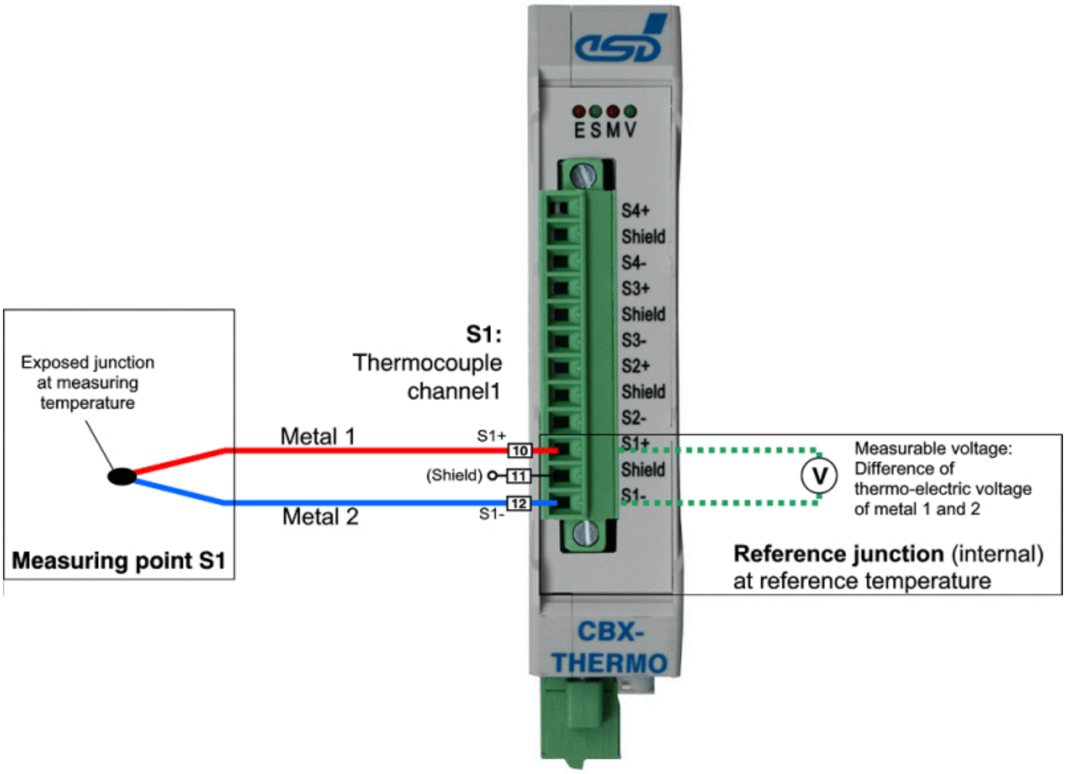


Fig. 3: Connection of Thermocouple S1

NOTICE
It is recommended to use a cable housing for the connection of the thermocouples (see Fig. 4), to minimize the error of the cold junction compensation (for order information see page 122).

NOTICE
Generally a standard thermocouple (length < 3 m) comes only with two signal lines.
If a shielded thermocouple extension cable is used, it should be connected to the shield potential of the corresponding channel.

INFORMATION
The connector pin assignment can be found on page 29 and following.
For conductor connection and conductor cross section see page 34.

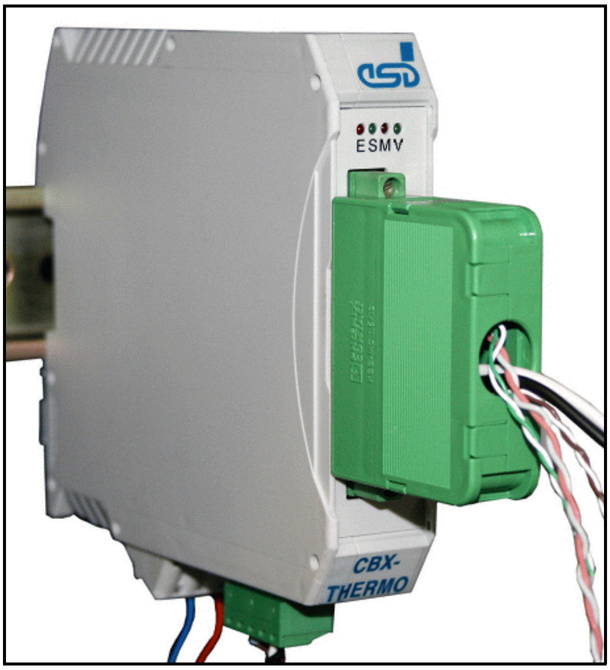


Fig. 4: Connection with cable housing to improve cold junction compensation

4.2 LED Display

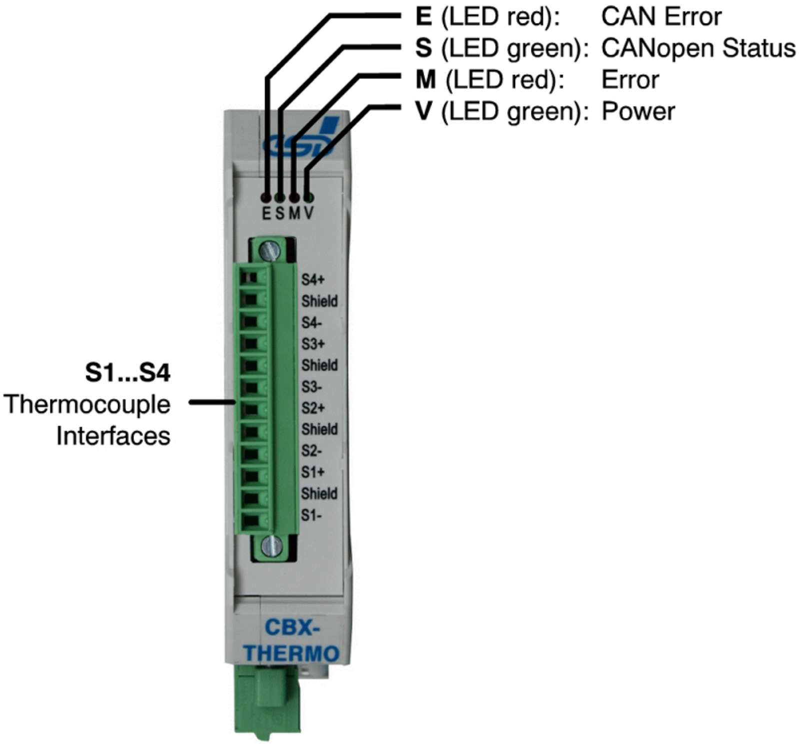


Fig. 5: Position of the LEDs in the front panel

The CAN-CBX-THERMO module is equipped with 4 status LEDs.
The terms of the indicator states of the LEDs are chosen in accordance with the terms recommended by the CiA [3]. The indicator states are described in the following chapters.

4.2.1 Indicator States

In principle there are 8 indicator states distinguished:

Indicator state	Display
on	LED constantly on
off	LED constantly off
blinking	LED blinking with a frequency of approx. 2.5 Hz
flickering	LED flickering with a frequency of approx. 10 Hz
1 flash	LED 200 ms on, 1400 ms off
2 flashes	LED 200 ms on, 200 ms off, 200 ms on 1000 ms off
3 flashes	LED 2x (200 ms on, 200 ms off) + 1x (200 ms on, 1000 ms off)
4 flashes	LED 3x (200 ms on, 200 ms off) + 1x (200 ms on, 1000 ms off)

Table 5: Indicator states

INFORMATION
Red and green LEDs are strictly switched in phase opposition according to the CANopen Specification [3].
For certain indicator states viewing all LEDs together might lead to a misinterpretation of the indicator states of adjacent LEDs. It is therefore recommended to look at the indicator state of an LED individually, in covering the adjacent LEDs.

4.2.2 Operation of the CAN-Error LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
E	CAN Error	red	off	no error
			1 flash	CAN controller is in *Error Active* state
			on	CAN controller state is *Bus Off* (or coding switch position ID-node > $7F_h$ when switching on; see 'Special Indicator States'on page 22)
			2 flashes	Heartbeat or Nodeguard error occurred. The LED automatically turns off, if Nodeguard/Heartbeat-messages are received again.

Table 6: Indicator states of the red CAN Error-LED

4.2.3 Operation of the CANopen-Status LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
S	CANopen Status	green	blinking	*Pre-operational*
			on	*Operational*
			1 flash	*Stopped*
			3 flashes	Module is in bootloader mode, the power LED is off, (or coding switch position ID-node > $7F_h$ when switching on; see page 22)

Table 7: Indicator states of the CANopen Status-LED

4.2.4 Operation of the Error-LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
M	Error	red	off	no error
			on	CAN Overrun Error The sample rate is set too high, therefore the firmware is not able to transmit all data on the CAN bus.
			2 flashes	Internal software error e.g.: - stored data have an invalid checksum therefore default values are loaded - internal watchdog has triggered - indicator state is continued until the module resets or an error occurs at the outputs.
			blinking	Sensor error (emergency error code 5030_h) e.g.: - sensor not connected - sensor data faulty

Table 8: Indicator state of the Error-LED

4.2.5 Operation of the Power-LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
V	Power	green	off	no power supply voltage; or the module is in Bootloader-Mode, this state is indicated by the CANopen status-LED (3 Flashes)
			on	power supply voltage is on and application software is running

Table 9: Indicator state of the Power-LED V

4.2.6 Special Indicator States

The special indicator state described in the following table is indicated by the CANopen-Status-LED and the CAN-Error-LED together:

LED indication	Description
- CANopen-Status LED: 3 flashes and - CAN-Error LED: on	Invalid nodeID: The coding switches for the Node-ID are set to an invalid ID-value, when switching on. The firmware application will be stopped.

Table 10: Special Indicator States

4.2.7 Assignment of the LED Labelling to the Name in the Schematic Diagram

Labelling on the CAN-CBX-THERMO	Name in the Schematic Diagram *1)
E	LED290A
S	LED290B
M	LED290C
P	LED290D

*1) The schematic diagram is not part of this manual.

4.3 Coding Switches

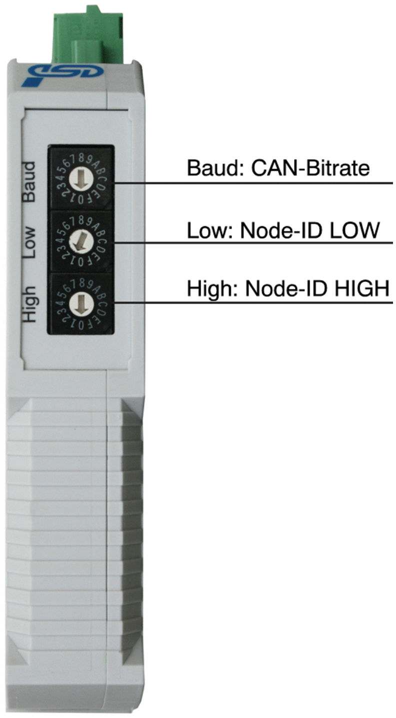


Fig. 6: Position of the coding switches

NOTICE
At the moment the module is switched ‘on’, the state of the coding switches is determined. Changes of the settings therefore have to be made **before switching on** the module, because changes of the settings are not determined during operation.

After a reset (e.g. NMT reset) the settings are read again.

4.3.1 Setting the Node-ID via Coding Switch

The address range of the CAN-CBX-module can be set *decimal* from 1 to 127 or *hexadecimal* from 01_h to $7F_h$.
The three higher-order bits (higher-order nibble) can be set with coding switch **HIGH**, the four lower-order bits can be set with coding switch **LOW**.

INFORMATION
Avoid the following settings:
Setting the address range of the coding switches to values higher than $7F_h$ causes error messages, the red CAN-Error LED is on.
If the coding switches are set to 00_h, the CAN-CBX-module changes into Bootloader mode.

4.3.2 Setting the Baud Rate

The baud rate can be set with the coding switch **Baud**.

Values from 0_h to F_h can be set via the coding switch. The values of the baud rate can be taken from the following table:

Setting	Bit rate [Kbit/s]
0	1000
1	$666,\overline{6}$
2	500
3	$333,\overline{3}$
4	250
5	166
6	125
7	100
8	$66,\overline{6}$
9	50
A_h	$33,\overline{3}$
B_h	20
C_h	12,5
D_h	10
E_h	800
F_h	$83,\overline{3}$

Table 11: Index of the baud rate

4.3.3 Assignment of Coding-Switch Labelling to Name in Schematic Diagram

Labelling on the CAN-CBX-THERMO	Name in the Schematic Diagram *1)
Baud	SW331
Low	SW330
High	SW332

*1) The Schematic Diagram is not part of this manual.

4.4 Installation of the Module Using InRailBus Connector

If the CAN bus signals and the power supply voltage shall be fed via the InRailBus, please proceed as follows:

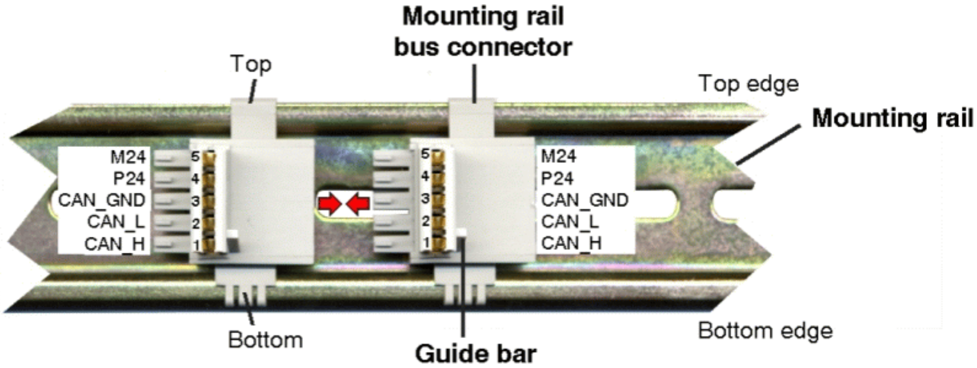


Figure 7: Mounting rail with bus connector

1. Position the InRailBus connector on the mounting rail and snap it onto the mounting rail using slight pressure. Plug the bus connectors together to contact the communication and power signals (in parallel with one). The bus connectors can be plugged together before or after mounting the CAN-CBX modules.

2. Place the CAN-CBX module with the DIN rail guideway on the top edge of the mounting rail.

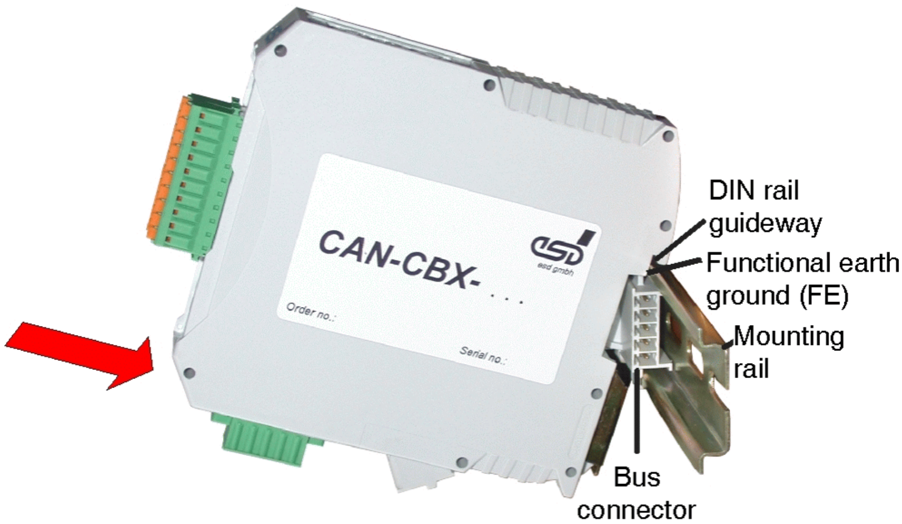


Figure 8 : Mounting CAN-CBX modules

3. Swivel the CAN-CBX module onto the mounting rail in pressing the module downwards according to the arrow as shown in figure 8. The housing is mechanically guided by the DIN rail bus connector.

4. When mounting the CAN-CBX module the metal foot catch snaps on the bottom edge of the mounting rail. Now the module is mounted on the mounting rail and connected to the InRailBus via the bus connector. Connect the bus connectors and the InRailBus if not already done.

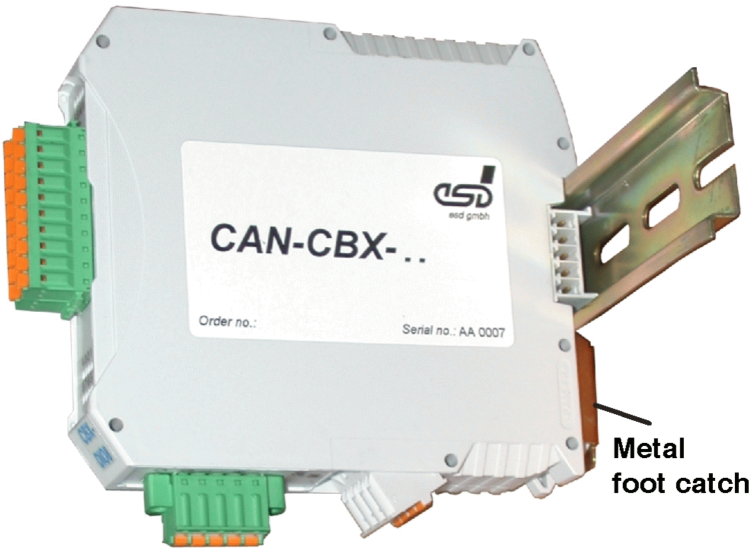


Figure 9: Mounted CAN-CBX module

4.4.1 Connecting Power Supply and CAN-Signals to CBX-InRailBus

To connect the power supply and the CAN-signals via the InRailBus, a terminal plug is needed. The terminal plug is not included in delivery and must be ordered separately (order no.: C.3000.02, see order information).

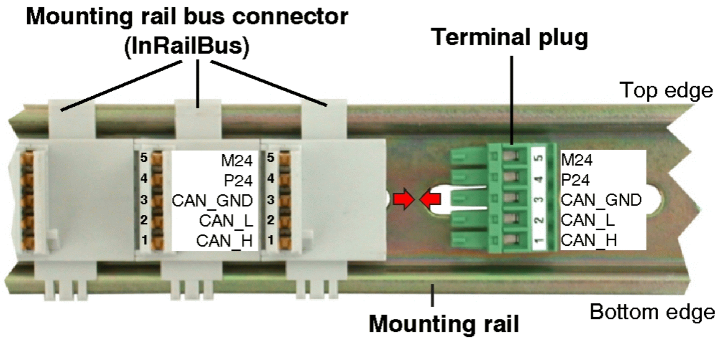


Fig. 10: Mounting rail with InRailBus and terminal plug

Plug the terminal plug into the socket on the right of the mounting-rail bus connector of the InRailBus, as described in Fig. 10. Then connect the CAN interface and the power supply voltage via the terminal plug.

4.4.2 Connection of the Power Supply Voltage

The power supply voltage can be supplied via the 24V connector or via the InRailBus.

	NOTICE Read and follow the safety instructions containing the requirements on power supply current circuits (see page 5)!

	NOTICE The connections for the 24 V power supply are internally connected and must **not** be supplied by two independent power sources at the same time!

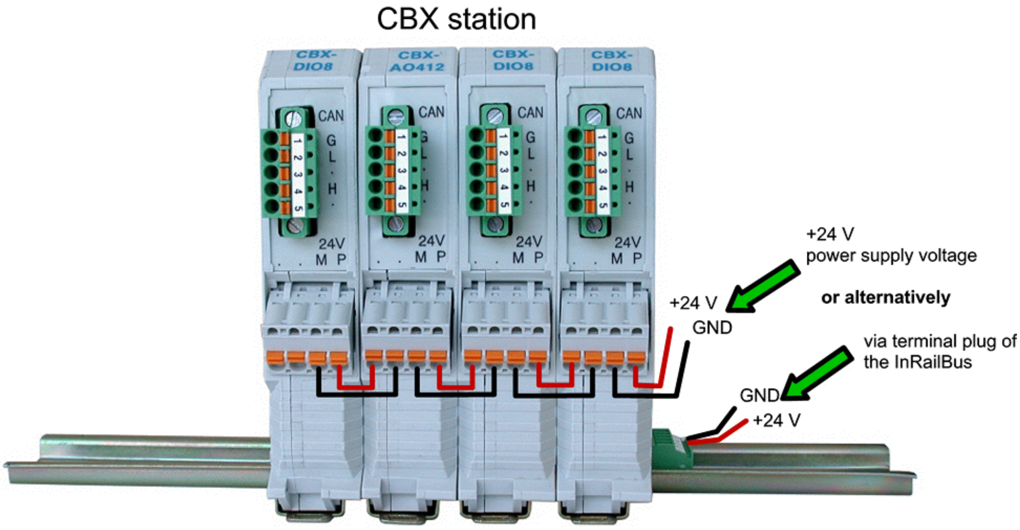


Fig. 11: Connecting the power supply voltage to the CAN-CBX station

Earthing of the Mounting Rail

	NOTICE The module is connected with the mounting rail via its functional earth contact. This improves the stability against electromagnetic disturbances. Thus the mounting rail shall be connected to an appropriate functional earth contact in the environment or in the installation. Please note that the impedance of the connection has to be kept low. The functional earth contact of the module does not ensure electrical safety.

4.4.3 Connection of CAN

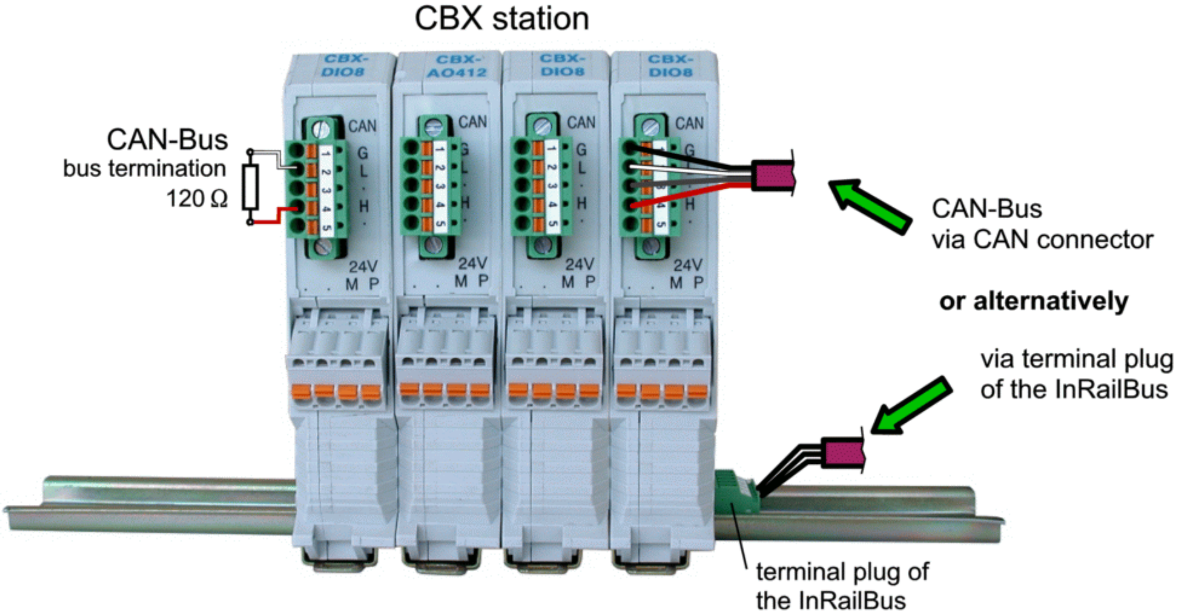


Fig. 12: Connecting the CAN signals to the CAN-CBX station

Generally the CAN signals can be fed via the CAN connector of the first CAN-CBX module of the CBX station. The signals are then connected through the CAN-CBX station via the InRailBus. To lead through the CAN signals through the CBX station the CAN bus connector of the last CAN-CBX module of the CAN-CBX station has to be used. The CAN connectors of the CAN-CBX modules which are not at the ends of the CAN-CBX station must not be connected to the CAN bus, because this would cause incorrect branching.
A bus termination must be connected to the CAN connector of the CAN-CBX module at the end of the CBX-InRailBus (see Fig. 12), if the CAN bus ends there.

4.5 Remove the CAN-CBX Module from the InRailBus

If the CAN-CBX module is connected to the InRailBus please proceed as follows:

Release the module from the mounting rail in moving the foot catch (see Fig. 9) downwards (e.g. with a screwdriver). Now the module is detached from the bottom edge of the mounting rail and can be removed.

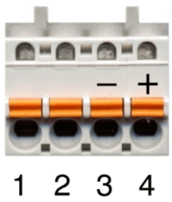
INFORMATION
It is possible to remove individual devices from the CBX station without interrupting the InRailBus connection, because the contact chain will not be disrupted.

5. Connector Assignment

5.1 Power Supply Voltage 24 V (X100)

Device connector: Phoenix-Contact MSTBO 2,5/4-G1L-KMGY
Line connector: Phoenix-Contact FKCT 2,5/4-ST, 5.0 mm pitch, spring-cage connection,
Phoenix-Contact order no.: 19 21 90 0 (included in the scope of delivery)
For conductor connection and conductor cross section see page 34.

Pin Position:



Pin Assignment:

Labelling on Housing	24V			
	•	•	M	P
Labelling on connector	(free)	(free)	-	+

Pin No.	1	2	3	4
Signal	P24 (+ 24 V)	M24 (GND)	M24 (GND)	P24 (+ 24 V)

Please refer also to the connecting diagram on page 16.

INFORMATION
The pins 1 and 4 are connected internally.
The pins 2 and 3 are connected internally.

Signal Description:

P24... power supply voltage +24 V
M24... reference potential

5.2 CAN

5.2.1 CAN Interface

The physical layer is designed according to ISO 11898-2. The CAN bus signals are electrically isolated from the other signals via a digital isolator and a DC/DC converter.

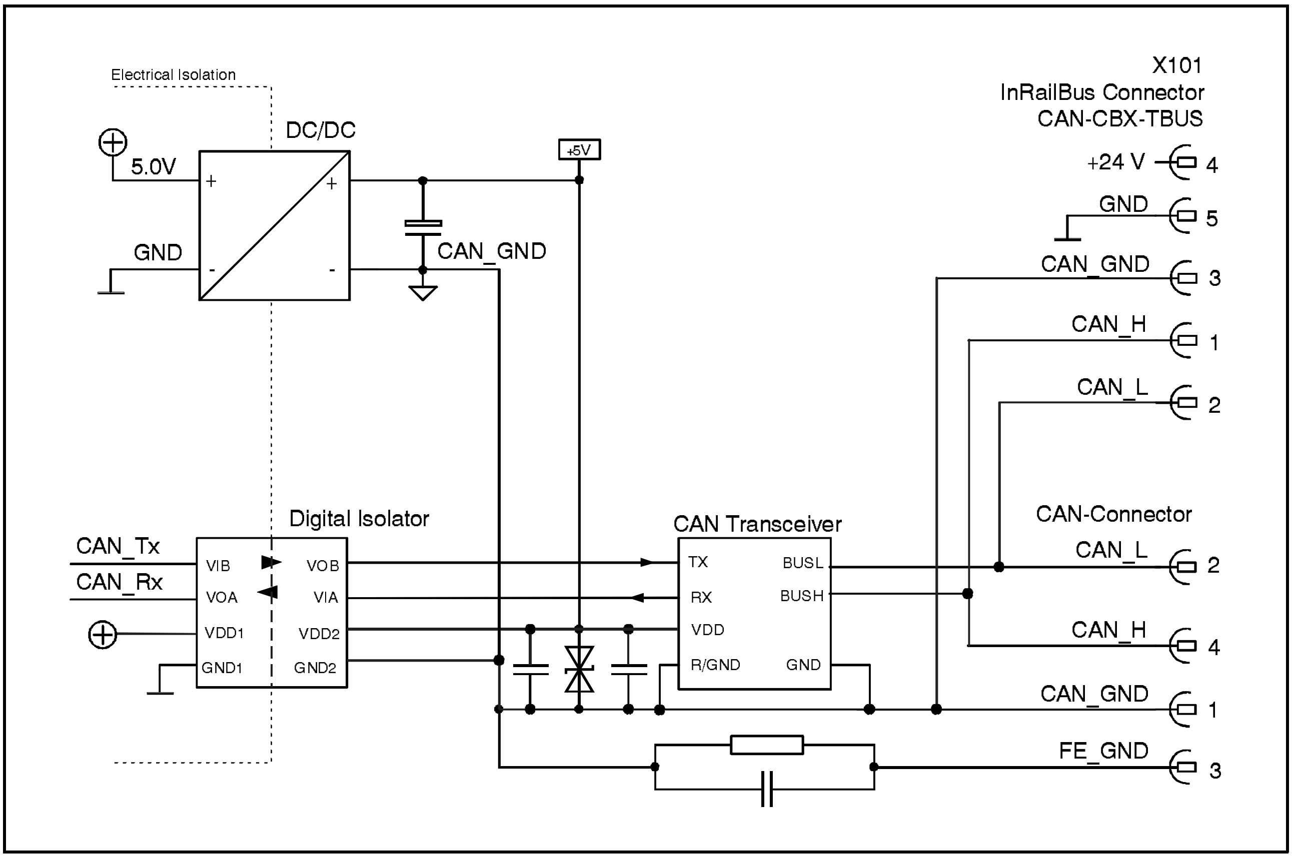


Fig. 13: CAN Interface

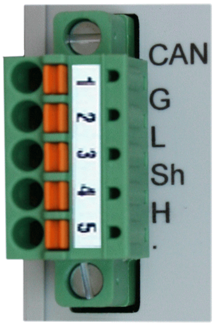
The CAN interface can be connected via the CAN connector or optionally via the InRailBus. Use the mounting-rail bus connector of the CBX-InRailBus (CAN-CBX-TBUS), see order information (page 122).

5.2.2 CAN Connector

Device Connector: Phoenix-Contact MC 1,5/5-GF-3,81
Line Connector: Phoenix-Contact FK-MCP 1,5/5-STF-3,81, spring-cage connection,
Phoenix-Contact order no.:1851261 (included in the scope of delivery)
For conductor connection and conductor cross section see page 34.

Pin Position:

(device connector with labelling)



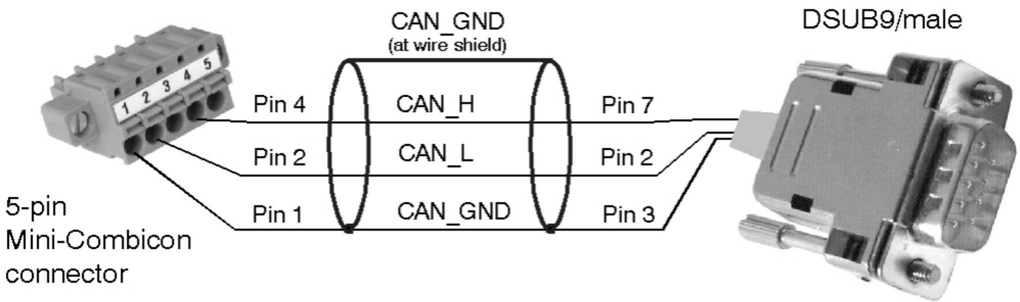
Pin-Assignment:

Labelling	Signal	Pin
G	CAN_GND	1
L	CAN_L	2
Sh	Shield	3
H	CAN_H	4
•	-	5

Signal description:

CAN_L, CAN_H ... CAN signals
CAN_GND ... reference potential of the local CAN physical layer
Shield ... pin for line shield connection (using hat rail mounting direct contact to the mounting rail potential)
- ... not connected

Recommendation of an adapter cable from 5-pin Phoenix Contact connector (here line connector FK-MCP1,5/5-STF-3,81 with spring-cage-connection) to 9-pin DSUB:

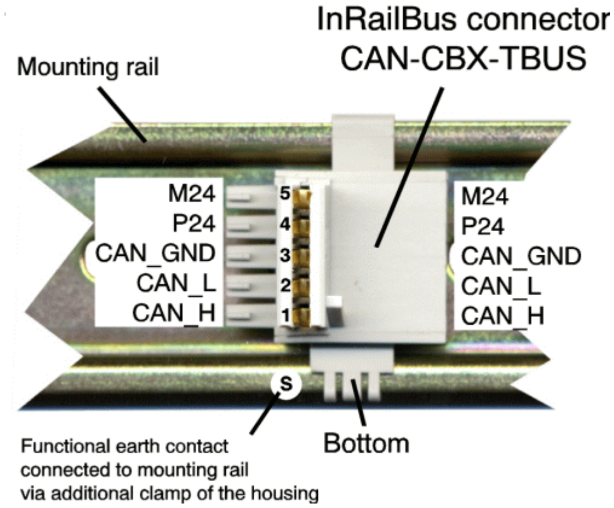


The 9-pin DSUB connector is assigned according to CiA 102

5.2.3 CAN and Power Supply Voltage via InRailBus Connector

Connector type: Mounting rail bus connector CAN-CBX-TBUS
(Phoenix-Contact ME 22,5 TBUS 1,5/5-ST-3,81 KMGY)

Pin Position:



Pin Assignment:

Pin	Signal
5	M24 (GND)
4	P24 (+24 V)
3	CAN_GND
2	CAN_L
1	CAN_H

S	FE (PE_GND)

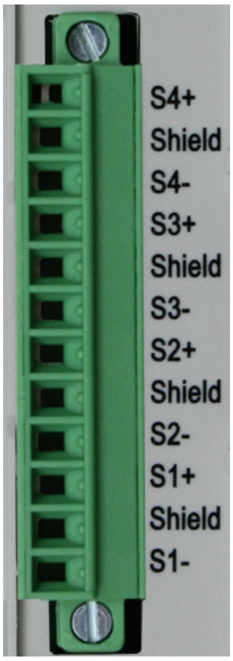
Signal Description:

CAN_L,
CAN_H ... CAN signals
CAN_GND ... reference potential of the local CAN-Physical layers
P24... power supply voltage +24 V
M24... reference potential
FE... functional earth contact (EMC)(connected to mounting rail potential)

5.3 Thermocouple Interface (X500)

Device's socket: Phoenix Contact MC 1,5/12-GF-3,81
Line connector: Phoenix Contact MC 1,5/12-STF-3,81, screw connection
Phoenix Contact order No.: 1827800 (included in delivery)
For conductor connection and conductor cross section see page 34.

Pin Position:



Pin Assignment:

Signal	Pin	Description
S4+	1	
Shield	2	Interface thermocouple 4
S4-	3	
S3+	4	
Shield	5	Interface thermocouple 3
S3-	6	
S2+	7	
Shield	8	Interface thermocouple 2
S2-	9	
S1+	10	
Shield	11	Interface thermocouple 1
S1-	12	

Signal description:

SX+, SX-... Signal lines of the thermocouple interface X (X = 1...4)
Shield ... Shield potential (connected to functional earth contact of the module)

INFORMATION
An example for the connection of a thermocouple is given on page 17

5.4 Conductor Connection/Conductor Cross Sections

The following table contains an extract of the technical data of the line connectors.

Interface	Power Supply Voltage 24 V [7]	CAN Connector [8]	Thermocouple Interface [8]
Connector type plug component (Range of articles)	FKCT 2,5/4-ST KMGY	FK-MCP 1,5/5-STF-3,81	MC 1,5/..-STF-3,81
Connection method	spring-cage connection	spring-cage connection	screw connection
Stripping length	10 mm	9 mm	7 mm
Conductor cross section solid min.	0.2 mm²	0.14 mm²	0,14 mm²
Conductor cross section solid max.	2.5 mm²	1.5 mm²	1,5 mm²
Conductor cross section stranded min.	0.2 mm²	0.14 mm²	0,14 mm²
Conductor cross section stranded max.	2.5 mm²	1.5 mm²	1,5 mm²
Conductor cross section stranded, with ferrule without plastic sleeve min.	0.25 mm²	0.25 mm²	0,25 mm²
Conductor cross section stranded, with ferrule without plastic sleeve max.	2.5 mm²	1.5 mm²	1,5 mm²
Conductor cross section stranded, with ferrule with plastic sleeve min.	0.25 mm²	0.25 mm²	0,25 mm²
Conductor cross section stranded, with ferrule with plastic sleeve max.	2.5 mm²	0.5 mm²	0,5 mm²
Conductor cross section AWG/kcmil min.	24	26	28
Conductor cross section AWG/kcmil max	12	16	16
2 conductors with same cross section, solid min.	n.a.	n.a.	0,08 mm²
2 conductors with same cross section, solid max.	n.a.	n.a.	0,5 mm²
2 conductors with same cross section, stranded min.	n.a.	n.a.	0,08 mm²
2 conductors with same cross section, stranded max.	n.a.	n.a.	0,75 mm²
2 conductors with same cross section, stranded, ferrules without plastic sleeve, min.	n.a.	n.a.	0,25 mm²
2 conductors with same cross section, stranded, ferrules without plastic sleeve, max.	n.a.	n.a.	0,34 mm²
2 conductors with same cross section, stranded, TWIN ferrules with plastic sleeve, min.	0.5 mm²	n.a.	0,5 mm²
2 conductors with same cross section, stranded, TWIN ferrules with plastic sleeve, max.	1 mm²	n.a.	0,5 mm²
Minimum AWG according to UL/CUL	26	28	30
Maximum AWG according to UL/CUL	12	16	14

n.a. ... not allowed

6. Correct Wiring of Electrically Isolated CAN Networks

For the CAN wiring all applicable rules and regulations (EC, DIN), e.g. regarding electromagnetic compatibility, security distances, cable cross-section or material, have to be met.

6.1 Standards concerning CAN Wiring

The flexibility in CAN network design is one of the key strengths of the various extensions and additional standards like e.g. CANopen, ARINC825, DeviceNet and NMEA2000 that have been built on the original ISO 11898-2 CAN standard. In using this flexibility comes the responsibility of good network design and balancing these tradeoffs.

Many CAN organizations and standards have scaled the use of CAN for applications outside the original ISO 11898. They have made system level tradeoffs for data rate, cable length, and parasitic loading of the bus.

However for CAN network design margin must be given for signal loss across the complete system and cabling, parasitic loadings, network imbalances, ground offsets against earth potential and signal integrity. **Therefore the practical maximum number of nodes, bus length and stub length are typically much lower.**

esd has concentrated her recommendations concerning CAN wiring on the specifications of the ISO 11898-2. Thus this wiring hints forgoes to describe the special features of the derived standards CANopen, ARINC825, DeviceNet and NMEA2000.

The consistent compliance to ISO 11898-2 offers significant advantages:

- Durable operation due to well proven design specifications
- Minimizing potential failures due to sufficient margin to physical limits
- Trouble-free maintenance during future network modifications or during fault diagnostics due to lack of exceptions

Of course reliable networks can be designed according to the specifications of CANopen, ARINC825, DeviceNet and NMEA2000, **however it must be observed that it is strictly not recommended to mix the wiring guidelines of the various specifications!**

6.2 Light Industrial Environment (Single Twisted Pair Cable)

6.2.1 General Rules

NOTICE
esd grants the EU Conformity of the product, if the CAN wiring is carried out with at least single shielded single twisted pair cables that match the requirements of ISO 118982-2. Single shielded *double* twisted pair cable wiring as described in chapter 6.3 ensures the EU Conformity as well.

The following **general rules** for CAN wiring with single shielded single twisted pair cable should be followed:

1	A cable type with a wave impedance of about 120 Ω ±10% with an adequate conductor cross-section (> 0.22 mm²) has to be used. The voltage drop over the wire has to be considered!
2	For light industrial environment use at least a **two-wire** CAN cable. Connect ● the two twisted wires to the data signals (CAN_H, CAN_L) and ● the cable shield to the reference potential (CAN_GND).
3	The reference potential CAN_GND has to be connected to the functional earth (FE) at exactly one point.
4	A CAN net must not branch (exception: short cable stubs) and has to be terminated with the characteristic impedance of the line (generally 120 Ω ±10%) at both ends (between the signals CAN_L and CAN_H and **not** at CAN_GND)!
5	Keep cable stubs as short as possible (l < 0.3 m)!
6	Select a working combination of bit rate and cable length.
7	Keep away cables from disturbing sources. If this cannot be avoided, double shielded wires are recommended.

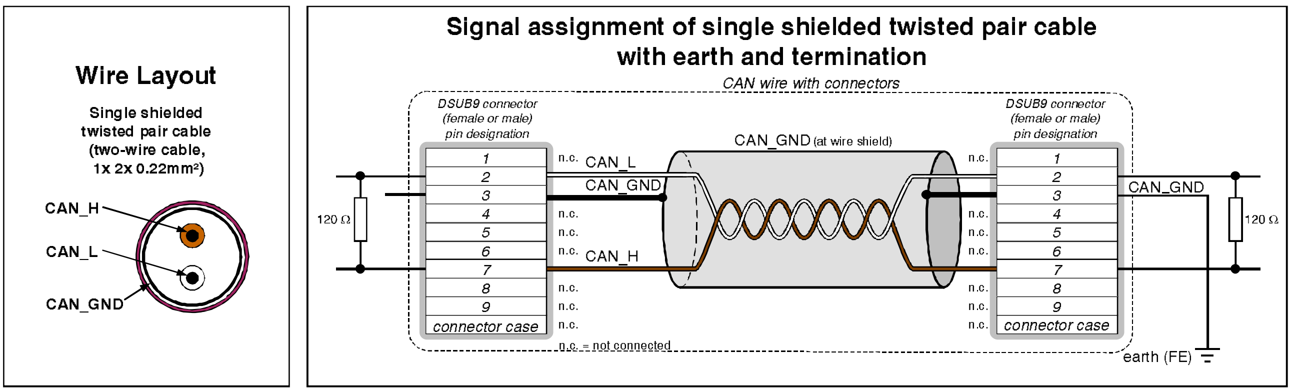


Figure. 14: CAN wiring for light industrial environment

6.2.2 Cabling

- To connect CAN devices with just one CAN connector per net use a short stub (< 0.3 m) and a T-connector (available as accessory). If this devices are located at the end of the CAN network, the CAN terminator "CAN-Termination-DSUB9" can be used.

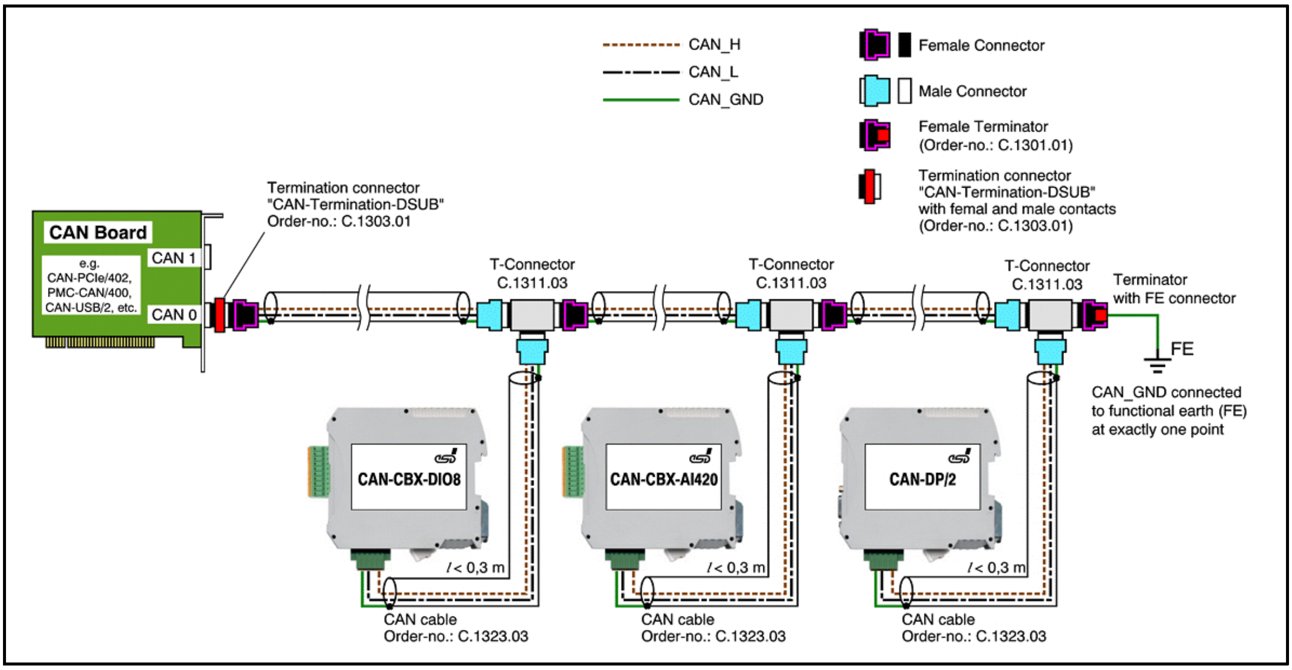


Figure. 15: Example for proper wiring with single shielded single twisted pair wires

6.2.3 Branching

- In principle the CAN bus has to be realized in a line. The participants are connected to the main CAN bus line via short cable stubs. This is normally realised by so called T-connectors. esd offers the CAN-T-Connector (Order No.: C.1311.03)
- If a mixed application of single twisted and double twisted cables is unavoidable, take care that the CAN_GND line is not interrupted!
- Deviations from the bus structure can be realized by the usage of repeaters.

6.2.4 Termination

- A termination resistor has to be connected at both ends of the CAN bus.
  If an integrated CAN termination resistor which is equipped at the CAN interface at the end of the bus is connected, this one has to be used for termination instead of an external CAN termination plug.
- 9-pin DSUB-termination connectors with integrated termination resistor and male and female contacts are available from esd (order no. C.1303.01).
- DSUB termination connectors with male contacts (order no. C.1302.01) or female contacts (order no. C.1301.01) and additional functional earth contact are available, if CAN termination <u>and grounding</u> of CAN_GND is required.

6.3 Heavy Industrial Environment (Double Twisted Pair Cable)

6.3.1 General Rules

The following **general rules** for CAN wiring with single shielded *double* twisted pair cable should be followed:

1	A cable type with a wave impedance of about 120 Ω ±10% with an adequate conductor cross section (> 0.22 mm²) has to be used. The voltage drop over the wire has to be considered.
2	For heavy industrial environment use a **four-wire** CAN cable. Connect ● two twisted wires to the data signals (CAN_H, CAN_L) and ● the other two twisted wires to the reference potential (CAN_GND) and ● the cable shield to functional earth (FE) at least at one point.
3	The reference potential CAN_GND has to be connected to the functional earth (FE) at exactly **one** point.
4	A CAN bus line must not branch (exception: short cable stubs) and has to be terminated with the characteristic impedance of the line (generally 120 Ω ±10%) at both ends (between the signals CAN_L and CAN_H and **not** to CAN_GND).
5	Keep cable stubs as short as possible (l < 0.3 m).
6	Select a working combination of bit rate and cable length.
7	Keep away CAN cables from disturbing sources. If this cannot be avoided, double shielded cables are recommended.

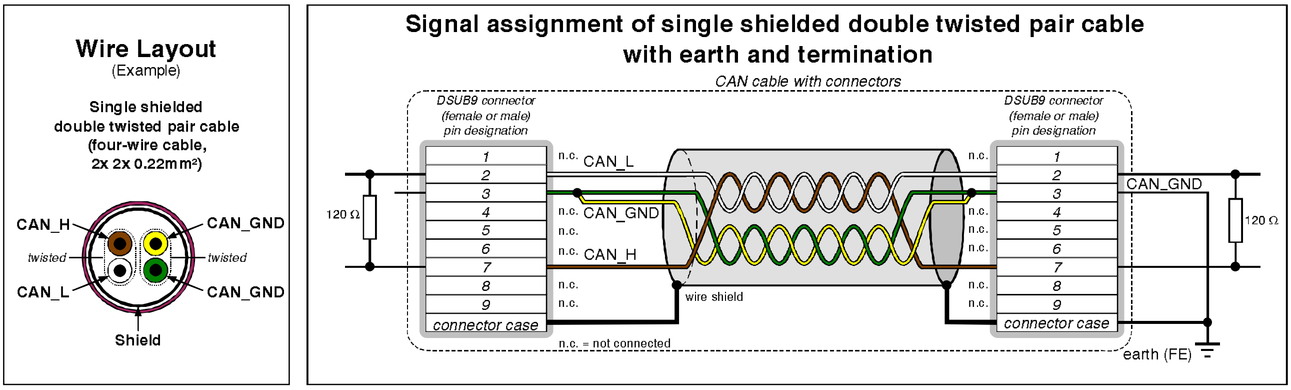


Fig. 16: CAN wiring for heavy industrial environment

6.3.2 Device Cabling

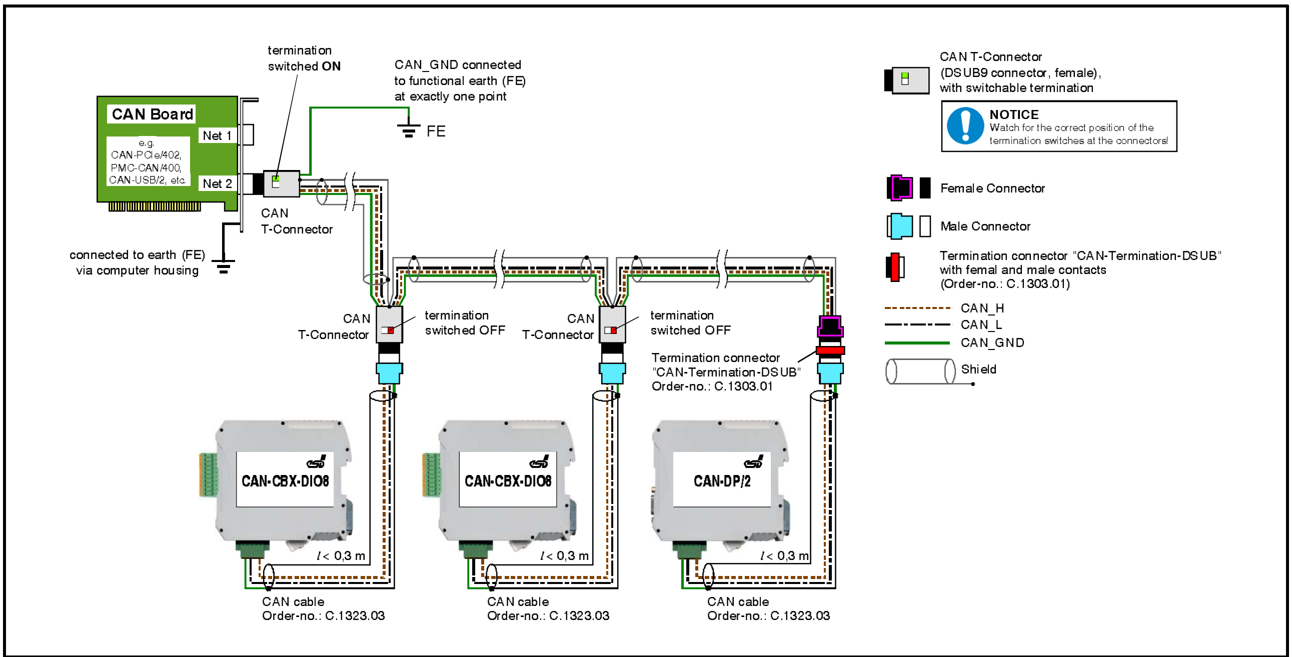


Fig. 17: Example for proper wiring with single shielded double twisted pair cables

6.3.3 Branching

- In principle the CAN bus has to be realized in a line. The participants are connected to the main CAN bus line via short cable stubs. This is normally realised by so called T-connectors. When using esd's CAN-T-Connector (order no.: C.1311.03) it should be noted that the shield potential of the conductive DSUB housing is not looped through this T-Connector type. Thus the shielding is interrupted. Therefore you have to take adequate measures to connect the shield potentials, as described in the manual of the CAN-T-Connector. For further information on this read the CAN-T-Connector Manual (order no.: C.1311.21).
  Alternatively a T-connector can be used, in which the shield potential is looped through, e.g. the DSUB9 connector from ERNI (ERBIC CAN BUS MAX, order no.:154039).
- If a mixed application of single twisted and double twisted cables is unavoidable, take care that the CAN_GND line is not interrupted!
- Deviations from the bus structure can be realized by the usage of repeaters.

6.3.4 Termination

- A termination resistor has to be connected at both ends of the CAN bus. If an integrated CAN termination resistor which is equipped at the CAN interface at the end of the bus is connected, this one has to be used for termination instead of an external CAN termination plug.
- 9-pin DSUB-termination connectors with integrated termination resistor and male and female contacts are available from esd (order no. C.1303.01).
- 9-pin DSUB-connectors with integrated switchable termination resistor can be ordered e.g. from ERNI (ERBIC CAN BUS MAX, female contacts, order no.:154039).

6.4 Electrical Grounding

- For CAN devices with electrical isolation the CAN_GND must be connected between the CAN devices.
- CAN_GND should be connected to the earth potential (FE) at **exactly one** point of the network.
- Each *CAN interface with electrical connection to earth potential* acts as grounding point.
  For this reason it is recommended not to connect more than one *CAN device with electrical connection to earth potential*.
- Grounding can be made e.g. at a termination connector (e.g. order no. C.1302.01 or C.1301.01)

6.5 Bus Length

NOTICE
Please note that the cables, connectors and termination resistors used in CANopen networks shall meet the requirements defined in ISO11898-2. In addition, further recommendations of the CiA, like standard values of the cross section, depending on the cable length, are described in the CiA recommendation CiA 303-1 (see CiA 303 CANopen Recommendation - Part 1: „Cabling and connector pin assignment“, Version 1.8.0, Table 2).

<table>
<tr><th>Bit-Rate [kBit/s]</th><th>Theoretical values of reachable wire length with esd interface l_{max} [m]</th><th>CiA recommendations (07/95) for reachable wire lengths l_{min} [m]</th><th>Standard values of cross-section according to CiA 303-1 [mm²]</th></tr>
<tr><td>1000</td><td>37</td><td>25</td><td>0.25 to 0.34</td></tr>
<tr><td>800</td><td>59</td><td>50</td><td rowspan="6">0.34 to 0.6</td></tr>
<tr><td>$666.\overline{6}$</td><td>80</td><td>-</td></tr>
<tr><td>500</td><td>130</td><td>100</td></tr>
<tr><td>$333.\overline{3}$</td><td>180</td><td>-</td></tr>
<tr><td>250</td><td>270</td><td>250</td></tr>
<tr><td>166</td><td>420</td><td>-</td></tr>
<tr><td>125</td><td>570</td><td>500</td><td rowspan="2">0.5 to 0.6</td></tr>
<tr><td>100</td><td>710</td><td>650</td></tr>
<tr><td>$83.\overline{3}$</td><td>850</td><td>-</td><td rowspan="4">0.75 to 0.8</td></tr>
<tr><td>$66.\overline{6}$</td><td>1000</td><td>-</td></tr>
<tr><td>50</td><td>1400</td><td>1000</td></tr>
<tr><td>$33.\overline{3}$</td><td>2000</td><td>-</td></tr>
<tr><td>20</td><td>3600</td><td>2500</td><td rowspan="4">not defined in CiA 303-1</td></tr>
<tr><td>12.5</td><td>5400</td><td>-</td></tr>
<tr><td>10</td><td>7300</td><td>5000</td></tr>
</table>

Table 12: Recommended cable lengths at typical bit rates (with esd-CAN interfaces)

- Optical couplers are delaying the CAN signals. esd modules typically reach a wire length of 37 m at 1 Mbit/s within a proper terminated CAN network without impedance disturbances like e.g. caused by cable stubs > 0.3 m.

6.6 Examples for CAN Cables

esd recommends the following two-wire and four-wire cable types for CAN network design. These cable types are used by esd for ready-made CAN cables, too.

6.6.1 Cable for Light Industrial Environment Applications (Two-Wire)

Manufacturer	Cable Type
U.I. LAPP GmbH Schulze-Delitzsch-Straße 25 70565 Stuttgart Germany www.lappkabel.de	e.g. UNITRONIC ®-BUS CAN UL/CSA (1x 2x 0.22) (UL/CSA approved) Part No.: 2170260 UNITRONIC ®-BUS-FD P CAN UL/CSA (1x 2x 0.25) (UL/CSA approved) Part No.: 2170272
ConCab GmbH Äußerer Eichwald 74535 Mainhardt Germany www.concab.de	e.g. BUS-PVC-C (1x 2x 0.22 mm²) Part No.: 93 022 016 (UL appr.) BUS-Schleppflex-PUR-C (1x 2x 0.25 mm²) Part No.: 94 025 016 (UL appr.)

6.6.2 Cable for Heavy Industrial Environment Applications (Four-Wire)

Manufacturer	Cable Type
U.I. LAPP GmbH Schulze-Delitzsch-Straße 25 70565 Stuttgart Germany www.lappkabel.de	e.g. UNITRONIC ®-BUS CAN UL/CSA (2x 2x 0.22) (UL/CSA approved) Part No: 2170261 UNITRONIC ®-BUS-FD P CAN UL/CSA (2x 2x 0.25) (UL/CSA approved) Part No.: 2170273
ConCab GmbH Äußerer Eichwald 74535 Mainhardt Germany www.concab.de	e.g. BUS-PVC-C (2x 2x 0.22 mm²) Part No.: 93 022 026 (UL appr.) BUS-Schleppflex-PUR-C (2x 2x 0.25 mm²) Part No.: 94 025 026 (UL appr.)

INFORMATION
Ready-made CAN cables with standard or custom length can be ordered from **esd.**

7. CAN Troubleshooting Guide

The CAN Troubleshooting Guide is a guide to find and eliminate the most frequent hardware-error causes in the wiring of CAN networks.

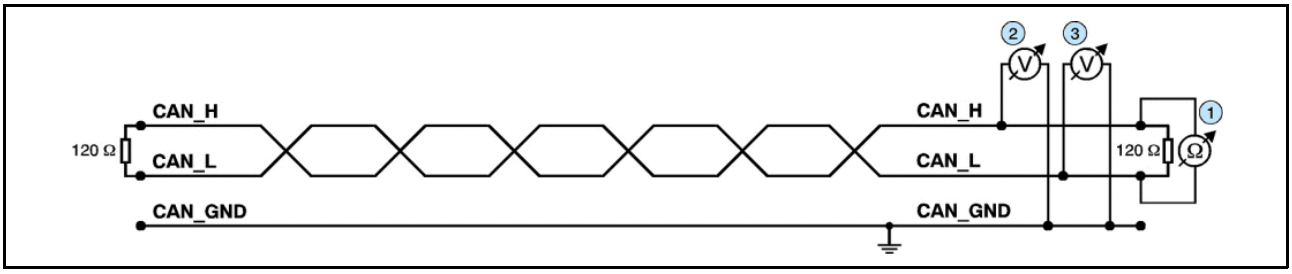


Figure. 18: Simplified diagram of a CAN network

7.1 Termination

The termination is used to match the impedance of a node to the impedance of the transmission line being used. When impedance is mismatched, the transmitted signal is not completely absorbed by the load and a portion is reflected back into the transmission line. If the source, transmission line and load impedance are equal these reflections are avoided. This test measures the series resistance of the CAN data pair conductors and the attached terminating resistors.

To test it, please

1. Turn off all power supplies of the attached CAN nodes.
2. Measure the DC resistance between CAN_H and CAN_L at one end of the network (1) (see figure above)

The measured value should be between 50 Ω and 70 Ω.

If the value is below 50 Ω, please make sure that:
- there is no **short circuit** between CAN_H and CAN_L wiring
- there are **not more than two** terminating resistors connected
- the nodes do not have faulty transceivers.

If the value is higher than 70 Ω, please make sure that:
- there are no open circuits in CAN_H or CAN_L wiring
- your bus system has two terminating resistors (one at each end) and that they are 120 Ω each.

7.2 Electrical Grounding

CAN_GND of the CAN network should be connected to Functional earth potential (FE) at only **one** point. This test will check if the CAN_GND is grounded in several places.
To test it, please

1. Disconnect the CAN_GND from the earth potential (FE).

2. Measure the DC resistance between CAN_GND and earth potential (see figure on the right).

3. Reconnect CAN_GND to earth potential.

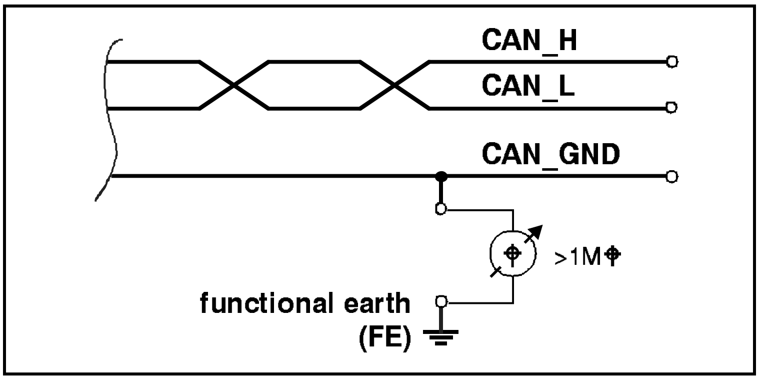


Fig. 19: Simplified schematic diagram of ground test measurement

The measured resistance should be higher than 1 M Ω. If it is lower, please search for additional grounding of the CAN_GND wires.

7.3 Short Circuit in CAN Wiring

A CAN bus might possibly still be able to transmit data if there is a short circuit between CAN_GND and CAN_L, but generally the error rate will increase strongly. Make sure that there is no short circuit between CAN_GND and CAN_L!

7.4 CAN_H/CAN_L Voltage

Each node contains a CAN transceiver that outputs differential signals. When the network communication is idle the CAN_H and CAN_L voltages are approximately 2.5 V measured to CAN_GND. Faulty transceivers can cause the idle voltages to vary and disrupt network communication.

To test for faulty transceivers, please

1. Turn on all supplies.
2. Stop all network communication.
3. Measure the DC voltage between CAN_H and CAN_GND (2) (see figure at previous page).
4. Measure the DC voltage between CAN_L and CAN_ GND (3) (see figure at previous page).

Normally the voltage should be between 2.0 V and 3.0 V.

If it is lower than 2.0 V or higher than 3.0 V, it is possible that one or more nodes have faulty transceivers.
For a voltage lower than 2.0 V please check CAN_H and CAN_L conductors for continuity.

To find the node with a faulty transceiver within a network please test the CAN transceiver resistance (see below) of the nodes.

7.5 CAN Transceiver Resistance Test

CAN transceivers have circuits that control CAN_H and CAN_L. Experience has shown that electrical damage of the circuits may increase the leakage current in these circuits.

To measure the current leakage through the CAN circuits, please use a resistance measuring device and:

1. Switch off the node and disconnect it from the network (4) (see figure below).
2. Measure the DC resistance between CAN_H and CAN_GND (5) (see figure below).
3. Measure the DC resistance between CAN_L and CAN_GND (6) (see figure below).

The measured resistance has to be about 500 kΩ for each signal. If it is much lower, the CAN transceiver is probably faulty. Another indication for a faulty transceiver is a very high deviation between the two measured input resistances (>> 200%).

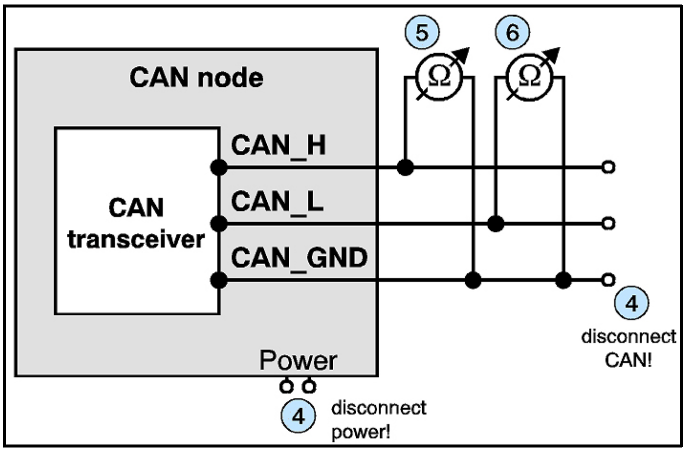


Figure 20: Simplified diagram of a CAN node

7.6 Support by esd

If you have executed the fault diagnostic steps of this troubleshooting guide and you even can not find a solution for your problem our support department will be able to assist.
Please contact our support via email at **support@esd.eu** or by phone **+40-511-37298-130**.

8. CANopen Firmware

Apart from basic descriptions of CANopen, this chapter contains the most significant information about the implemented functions.

A complete CANopen description is too extensive for the purpose of this manual.
Further information can therefore be taken from the CANopen documentation [1] and [4].

8.1 Definition of Terms

COB ...	Communication Object
Emergency-Id...	Emergency Data Object
NMT...	Network Management (Master)
SDO...	Service Data Object
Sync...	Sync(frame) Telegram

PDOs (Process Data Objects)

PDOs are used to transmit process data.
In the 'Transmit'-PDO (TPDO) the CAN-CBX-module transmits data to the CANopen network.
In the 'Receive'-PDO (RPDO) the CAN-CBX-module receives data from the CANopen network.

SDOs (Service Data Objects)

SDOs are used to transmit module internal configuration- and parameter data. In opposition to the PDOs SDO-messages are confirmed. A write or read request on a data object is always answered by a response telegram with an error index.

8.2 NMT-Boot-up

The CAN-CBX module can be initialized with the 'Minimum Capability Device' boot-up as described in [1].

Usually a telegram to switch from *Pre-Operational* status to *Operational* status after boot-up is sufficient. For this the 2-byte telegram '01_h', '00_h', for example, has to be transmitted with CAN-identifier '0000_h' (= Start Remote Node all Devices).

8.3 The CANopen-Object Directory

The object directory is basically a (sorted) group of objects which can be accessed via the CAN network. Each object in this directory is addressed with a 16-bit index. The index in the object directories is represented in hexadecimal format.
The index can be a 16-bit parameter in accordance with the CANopen specification [1] or a manufacturer-specific code. By means of the MSBs of the index the object class of the parameter is defined.

Part of the object directory are among others:

Index	Object
0001_h ... $009F_h$	definition of data types
1000_h ... $1FFF_h$	Communication Profile Area
2000_h ... $5FFF_h$	Manufacturer Specific Profile Area
6000_h ... $9FFF_h$	Standardized Device Profile Area
$A000_h$ ... $FFFF_h$	reserved

8.4 Communication Parameters of the PDOs

The communication parameters of the PDOs (according to [1]) are transmitted as SDO (Service Data Objects) on ID ‘**600_h + Node-ID**’ (Request). The receiver acknowledges the parameters on ID ‘**580_h + Node-ID**’ (Response).

The **Node-ID** (module No.) is configured via coding switches Low and High. Please refer to chapter “Coding Switches” (page 23) for a detailed description of possible configurations.

8.4.1 Access on the Object Directory

The SDOs (Service Data Objects) are used to access the object directory of a device.
An SDO is therefore a ‘channel’ to access the parameters of the device. Access via this channel is possible in *operational* and *pre-operational* status.

The SDOs (Service Data Objects) are transmitted on ID ‘**600_h + Node-ID**’ (request).
The server acknowledges the parameters on ID ‘**580_h + Node-ID**’ (response).

An SDO is structured as follows:

Identifier	Command code	Index		Sub-index	LSB	Data field		MSB
		(low)	(high)					

Example:

600_h+ Node-ID	23_h	00_h	14_h	01_h	$7F_h$	04_h	00_h	00_h
	(write)	(Index=1400_h) (Receive-PDO-Comm-Para)		(COB-def.)	COB Node ID = 0000 $047F_h$			

Identifier
The parameters are transmitted with ID ‘600_h + NodeID’ (request).
The receiver acknowledges the parameters with ID ‘580_h + NodeID’ (response).

Command code
The command code transmitted consists among other things of the Command Specifier and the length. Frequently required combinations are, for instance:

$40_h = 64_{dec}$: Read Request, i.e. a parameter is to be read
$23_h = 35_{dec}$: Write Request with 32-bit data, i.e. a parameter is to be set

The CAN-CBX-module responds to every received telegram with a response telegram. This can contain the following command codes:

$43_h = 67_{dec}$: Read Response with 32 bit data, this telegram contains the parameter requested
$60_h = 96_{dec}$: Write Response, i.e. a parameter has been set successfully
$80_h = 128_{dec}$: Error Response, i.e. the CAN-CBX-module reports a communication error

Frequently Used Command Codes
The following table summarizes frequently used command codes. The command frames must always contain 8 data bytes. Notes on the syntax and further command codes can be found in [1].

Command	Number of data bytes	Command code
Write Request (Initiate Domain Download)	1 2 3 4	$2F_h$ $2B_h$ 27_h 23_h
Write Response (Initiate Domain Download)	-	60_h
Read Request (Initiate Domain Upload)	-	40_h
Read Response (Initiate Domain Upload)	1 2 3 4	$4F_h$ $4B_h$ 47_h 43_h
Error Response (Abort Domain Transfer)	-	80_h

Index, Sub-Index
Index and sub-index will be described in the chapters “Device Profile Area” and “Manufacturer Specific Objects” of this manual.

Data Field
The data field has got a size of a maximum of 4 bytes and is always structured ‘LSB first, MSB last’. The least significant byte is always in ‘Data 1’. With 16-bit values the most significant byte (bits 8...15) is always in ‘Data 2’, and with 32-bit values the MSB (bits 24...31) is always in ‘Data 4’.

Error Codes of the SDO Transfer

The following error codes might occur (according to [1]):

Abort Code	Description
05040001_h	wrong command specifier
06010002_h	wrong write access
06020000_h	wrong index
06040041_h	object can not be mapped to PDO
06060000_h	access failed due to an hardware error
06070010_h	wrong number of data bytes
06070012_h	service parameter too long
06070013_h	service parameter too small
06090011_h	wrong sub-index
06090030_h	transmitted parameter is outside the accepted value range
08000000_h	undefined cause of error
08000020_h	data cannot be transferred or stored in the application
08000022_h	data cannot be transferred or stored in the application because of the present device state
08000024_h	access to flash failed

8.5 Overview of used CANopen-Identifiers

Function	Identifier	Description
Network management	0	NMT
SYNC	80_h	Sync to all, (configurable via object 1005_h)
Emergency Message	80_h + *Node-ID*	configurable via object 1014_h
TPDO1	180_h + *Node-ID*	PDO1 from CAN-CBX-THERMO (object 1800_h)
TPDO2	280_h + *Node-ID*	PDO2 from CAN-CBX-THERMO (object 1801_h)
TPDO3	380_h + *Node-ID*	PDO3 from CAN-CBX-THERMO (object 1802_h)
TPDO4	480_h + *Node-ID*	PDO4 from CAN-CBX-THERMO (object 1803_h)
Client SDO	580_h + *Node-ID*	SDO from CAN-CBX-THERMO
Server SDO	600_h + *Node-ID*	SDO to CAN-CBX-THERMO
Node Guarding	700_h + *Node-ID*	configurable via object $100E_h$

NodeID: CANopen address [1_h...$7F_h$]

8.5.1 Setting the COB-ID

The COB-IDs which can be set (except the one of SYNC), are deduced initially from the setting of the Node-ID via the coding switches (see page 23). If the COB-IDs are set via SDO, this setting is valid even if the coding switches are set to another Node-ID after that.
To accept the Node-ID from the coding switches again, the *Comm defaults* or all defaults have to be restored (object 1011_h)

8.6 Default PDO-Assignment

PDOs (Process Data Objects) are used to transmit process data. The PDO mapping can be changed. The following tables show the default mapping at delivery of the module:

PDO	CAN Identifier	Length	Transmission direction	Assignment
TPDO1	180_h + Node-ID	4 byte	from CAN-CBX-THERMO (Transmit PDO)	Process value 1 (9130_h, 1)
TPDO2	280_h + Node-ID	4 byte		Process value 2 (9130_h, 2)
TPDO3	380_h + Node-ID	4 byte		Process value 3 (9130_h, 3)
TPDO4	480_h + Node-ID	4 byte		Process value 4 (9130_h, 4)

Per default only the *Process_Values* 1 to 4 (object 9130_h, sub-index 1 - 4) are mapped. Additionally the *Field_Values* 1 to 4 (object 9100_h, sub-index 1 - 4) can be mapped (see page 82).

TPDO1 (CAN-CBX-THERMO ->)
CAN Identifier: 180_h + Node-ID

Byte	0	1	2	3
Parameter	*Process_Value_1*			

TPDO2 (CAN-CBX-THERMO ->)
CAN Identifier: 280_h + Node-ID

Byte	0	1	2	3
Parameter	*Process_Value_2*			

TPDO3 (CAN-CBX-THERMO ->)
CAN-Identifier: 380_h + Node-ID

Byte	0	1	2	3
Parameter	*Process_Value_3*			

TPDO4 (CAN-CBX-THERMO ->)
CAN Identifier: 480_h + Node-ID

Byte	0	1	2	3
Parameter	*Process_Value_4*			

8.7 Reading the Analog Values

8.7.1 Messages of the Analog Inputs

The transmission types for the analog inputs are described in the following:

- *acyclic, synchronous*: The transmission is initiated if a SYNC-message has been received (PDO-transmission type 0) and data has changed.

- *cyclic, synchronous*: The transmission is initiated if a defined number of SYNC-messages have been received (PDO-transmission type 1...240).

- *event controlled, asynchronous*: The transmission is initiated if the state of selected inputs has changed (PDO-transmission type 254, 255).

8.7.2 Supported Transmission Types Based on DS-301

Transmission Type	PDO-Transmission					supported by CAN-CBX-THERMO
	cyclic	acyclic	synchro-nous	asynchro-nous	RTR	
0		X	X			x
1...240	X		X			x
241...253	reserved					-
254				X		x
255				X		x

8.8 Communication Profile Area

8.8.1 Used Names and Abbreviations

The following names are used in the tables for the description of the communication parameters:

PDO-Mappable	PDO-Mapping is possible for this sub-index of the PDO
Save to EEPROM	the value of this parameter is stored in the local EEPROM, if the command 'save' is called (see page 66)
Data type	data type (e.g. unsigned 8, unsigned 32)
Access mode	allowed access modes to this parameter ro... read_only This parameter can only be read. Write accesses will cause an error message. const....constant This parameter can not be set by the user. It is readable. Write accesses will cause an error message. rw... read&write This parameter can be read or written.
Value range	value range of the parameter
Default value	default setting of the parameter
Name/Description	name and short description of the parameter

8.9 Implemented CANopen-Objects

A detailed description of the objects can be taken from CiA DS-301.

8.9.1 Overview of used Communication Profile Objects and Product-specific Values

Index	Sub-index (max.)	Description	Data type	Access	Product-specific values
1000_h	-	Device Type	unsigned 32	ro	00020194_h
1001_h	-	Error Register	unsigned 8	ro	supported error bits: 0: generic 4: communication error 5: device profile 7: manufacturer
1003_h	A_h	Pre-Defined-Error-Field	unsigned32	rw	default: 0
1005_h	-	COB-ID-Sync	unsigned32	rw	default: 80_h
1006_h	-	Communication Cycle Period	unsigned32	rw	default: 0
1008_h	-	Manufacturer Device Name	visible string	ro	default: “CAN-CBX-THERMO”
1009_h	-	Manufacturer Hardware Version	visible string	ro	default depending on version
$100A_h$	-	Manufacturer Software Version	visible string	ro	default depending on version
$100C_h$	-	Guard Time	unsigned 16	rw	default: 0
$100D_h$	-	Life Time Factor	unsigned 8	rw	default: 0
$100E_h$	-	Node Guarding Identifier	unsigned 32	rw	Node-ID + 700_h
1010_h	3	Store Parameter	unsigned 32	rw	-
1011_h	3	Restore Parameter	unsigned 32	rw	-
1014_h	-	COB-ID Emergency Object	unsigned 32	rw	default: 0 80_h + Node-ID
1015_h	-	Inhibit Time EMCY	unsigned 16	rw	default: 0
1016_h	1	Consumer Heartbeat Time	unsigned 32	rw	default: 0
1017_h	-	Producer Heartbeat Time	unsigned 16	rw	default: 0
1018_h	4	Identity Object	unsigned 32	ro	Vendor Id: 00000017_h Prod. Code: 23034002_h (= C.3034.02)
1019_h	-	Synchronous Counter Overflow	unsigned 8	rw	default: 0
1020_h	2	Verify Configuration	unsigned 32	rw	default: 0
1029_h	1	Error Behaviour	unsigned 8	ro	00_h

Index	Sub-index (max.)	Description	Data type	Access
1800_h	5	1. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
1801_h	5	2. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
1802_h	5	3. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
1803_h	5	4. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
$1A00_h$	2	1. Transmit PDO-Mapping	PDO Mappping (21_h)	rw
$1A01_h$	2	2. Transmit PDO-Mapping	PDO Mappping (21_h)	rw
$1A02_h$	2	3. Transmit PDO-Mapping	PDO Mappping (21_h)	rw
$1A03_h$	2	4. Transmit PDO-Mapping	PDO Mappping (21_h)	rw

Index	Sub-index (max.)	Description	Data type	Access	Product-specific values
$1F80_h$	-	NMT startup	unsigned 32	rw	default: 2 (autostart disabled)
$1F91_h$	1	Self starting nodes timing parameters	unsigned 16	rw	default: 64_h (= 100 ms)

8.9.2 Device Type (1000_h)

INDEX	1000_h
Name	*device type*
Data type	unsigned 32
Access mode	ro
Default value	see chapter 8.9.1 (page 54)

Example: Reading the Device Type

The CANopen master transmits the read request with identifier ‘603_h’ (600_h + Node-ID) to the CAN-CBX module with the module no. 3 (Node-ID=3_h):

ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
603_h	0_h	8_h	40_h	00_h	10_h	00_h	00_h	00_h	00_h	00_h
			Read Request	Index=1000_h		Sub Index				

The CAN-CBX module no. 3 responds to the client by means of read response with identifier ‘583_h’ (580_h + Node-ID) with the value of the device type:

ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
583_h	0_h	8_h	43_h	00_h	10_h	00_h	**94_h**	**01_h**	**02_h**	**00_h**
			Read Response	Index=1000_h		Sub Index	Example here: Device Profile Nr.0191_h		Example here: Input	

value of device type: 0002.0194_h.
The value of the device type of this CAN-CBX module is printed in chapter 8.9.1 (page 54)

The data field is always structured following the rule ‘LSB first, MSB last‘ (see page 48, data field).

8.9.3 Error Register (1001_h)

The CAN-CBX module uses the error register to indicate error messages.

INDEX	**1001_h**
Name	*error register*
Data type	unsigned 8
Access type	ro
Default value	0

The following bits of the error register are being supported at present:

Bit	Meaning
0	*generic*
1	*current*
2	*voltage*
3	*temperature*
4	*communication error* (overrun, error state)
5	*device profile*
6	reserved
7	*manufacturer*

For a list of the error bits supported by this CAN-CBX module see chapter 8.9.1 (page 54).

Bits which are not supported are always returned as ‘0’.
If an error is active, the according bit is set to ‘1’.

8.9.4 Pre-defined Error Field (1003_h)

INDEX	**1003_h**
Name	*pre-defined error field*
Data type	unsigned 32
Access mode	ro
Default value	No

The *pre-defined error field* provides an error history of the errors that have occurred on the device and have been signalled via the Emergency Object.
Sub-index 0 contains the current number of errors stored in the list.
Under sub-index 1 the last error which occurred is stored. If a new error occurs, the previous error is stored under sub-index 2 and the new error under sub-index 1, etc. In this way a list of the error history is created.
The error buffer is structured like a ring buffer. If it is full, the oldest entry is deleted for the latest entry.

This module supports a maximum of 10 error entries. When the 11th error occurs the oldest error entry is deleted. In order to delete the entire error list, sub-index ‘0’ has to be set to ‘0’. This is the only permissible write access to the object.

With every new entry to the list the module transmits an **Emergency Frame** to report the error.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1003_h	0	*no_of_errors_in_list*	0, 1...A_h	-	unsigned 8	rw
	1	*error-code n*	0...$FFFFFFFF_h$	-	unsigned 32	ro
	2	*error-code (n-1)*	0...$FFFFFFFF_h$	-	unsigned 32	ro
	:	:	:	:	:	ro
	A_h	*error-code (n-9)*	0...$FFFFFFFF_h$	-	unsigned 32	ro

Meaning of the variables:

no_of_errors_in_list - contains the number of error codes currently on the list
n = number of error which occurred last
- in order to delete the error list this variable has to be set to ‘0’
- if *no_of_errors_in_list* $\neq$0, the error register (Object 1001_h) is set

error-code x The 32-bit long error code consists of the CANopen-emergency error code described in [1] and the error code defined by esd (manufacturer-specific error field).

Bit:	31 16	15 0
Contents:	*manufacturer-specific error field*	*emergency-error-code*

manufacturer-specific error field: always ‘00’, unless *emergency-error-code* = 2300_h (see below)

emergency-error-code: The following error-codes are supported:

8110_h - CAN overrun error
- Sample rate is set too high, thus the firmware is not able to transmit all data to the CAN bus.

8120_h - CAN in error passive mode

8130_h - Lifeguard error / heartbeat error

8140_h - Recovered from “Bus Off”

8240_h - Unexpected SYNC data length

6000_h - Software error:
-EEPROM checksum error (no transmission of this error message as emergency message)

6110_h - Internal Software error
e.g.:
- saved data had invalid checksum and default data is loaded

$FF10_h$ - Data loss (A/D data overflow)

5000_h - Hardware error (e.g. A/D-converter defective)

5030_h - Sensor error

Emergency Message

The data otf the emergency frame transmitted by the CAN-CBX-module have the following structure:

Byte:	0	1	2	3	4	5	6	7
Contents:	*emergency-error-code* (siehe oben)		*error-register* 1001_h	*no_of_errors _ in_list* $1003,00_h$	-			

An emergency message is transmitted, if an error occurs. If this error occurs again, no further emergency message is generated.
If the last error message is cancelled, again an emergency message is transmitted to indicate the error disappearance.

8.9.5 COB-ID of SYNC-Message (1005_h)

INDEX	**1005_h**
Name	*COB-ID SYNC message*
Data type	unsigned 32
Access mode	rw
Default value	see chapter 8.9.1 (page 54)

Structure of the parameter:

Bit-No.	Value	Meaning
31 (MSB)	-	do not care
30	0/1	0: Device does not generate SYNC message 1: Device generates SYNC message
29	0	always 0 (11-bit ID)
28...11	0	always 0 (29-bit IDs are not supported)
10...0 (LSB)	x	Bit 0...10 of the SYNC-COB-ID

The identifier can take values between 0...$7FF_h$.

8.9.6 Communication Cycle Period (1006_h)

INDEX	**1006_h**
Name	*Communication Cycle Period*
Data type	unsigned 32
Access mode	rw
Default value	0 µs

Value range of the parameter:

Value	Meaning
0	No transmission of SYNC messages
$1...FFFFFFFF_h$	Cycle time in microseconds

8.9.7 Manufacturer Device Name (1008_h)

INDEX	**1008_h**
Name	*manufacturer device name*
Data type	visible string
Default value	see chapter 8.9.1 (page 54)

For detailed description of the SDO Uploads, please refer to [1].

8.9.8 Manufacturer Hardware Version (1009_h)

INDEX	**1009_h**
Name	*manufacturer hardware version*
Data type	visible string
Default value	string: e.g. ‘1.00’ (depending on version)

The hardware version is read similarly to reading the manufacturer’s device name via the domain upload protocol. Please refer to [1] for a detailed description of the upload.

8.9.9 Manufacturer Software Version ($100A_h$)

INDEX	**$100A_h$**
Name	*manufacturer software version*
Data type	visible string
Default value	string: e.g.: ‘1.2’ (depending on version)

Reading the software version is similar to reading the manufacturer’s device name via the domain upload protocol. Please refer to [1] for a detailed description of the upload.

8.9.10 Guard Time ($100C_h$) und Life Time Factor ($100D_h$)

The CAN-CBX module supports the node guarding or alternatively the heartbeat function (see page 74).

NOTICE
By the recommendation of the CiA, the heartbeat-function shall be used preferentially.
Use the node-guarding only for existing systems and not for new developments!

Guard time and life time factors are evaluated together. Multiplying both values will give you the life time. The guard time is represented in milliseconds.

INDEX	**$100C_h$**
Name	*guard time*
Data type	unsigned 16
Access mode	rw
Default value	0 [ms]
Minimum value	0
Maximum value	$FFFF_h$ (65.535 s)

INDEX	**$100D_h$**
Name	*life time factor*
Data type	unsigned 8
Access mode	rw
Default value	0
Minimum value	0
Maximum value	FF_h

8.9.11 Node Guarding Identifier ($100E_h$)

The module only supports 11-bit identifiers.

INDEX	**$100E_h$**
Name	*node guarding identifier*
Data type	unsigned 32
Access mode	rw
Default value	700_h + Node-ID

Structure of the parameter *node guarding identifier* :

Bit-No.	Meaning
31 (MSB) 30	reserved
29...11	always 0, because 29-bit-IDs are not supported
10...0 (LSB)	bit 0...10 of the node guarding identifier

The identifier can take values between 1...$7FF_h$.

8.9.12 Store Parameters (1010_h)

INDEX	**1010_h**
Name	*store parameters*
Data type	unsigned 32

This object supports saving of parameters to a non-volatile memory, the EEPROM here.
Therefore the parameter groups shown below are distinguished. After they are transferred, the parameters are immediately active. The non-volatile storage of the parameters however is not carried out automatically. It must be initiated with a write access to object 1010_h and should only be carried out if the module is in the state *pre-operational*. In order to avoid storage of parameters by mistake, storage is only executed when the specific signature as shown below is transmitted.

Reading the index returns information about the implemented storage functionality (refer to [1] for more information).

Index	Sub-index	Description	Value range	Data type	Access mode
1010_h	0	*number_of_entries*	4	unsigned 8	ro
	1	*save_all_parameters* (objects 1000_h ... $9FFF_h$)	no default, write: 65 76 61 73_h (= ASCII: ‘e’ ‘v’ ‘a’ ‘s’)	unsigned 32	rw
	2	*save_communication_parameter* (objects 1000_h ... $1FFF_h$)		unsigned 32	rw
	3	*save_application_parameter* (objects 6000_h ... $9FFF_h$)		unsigned 32	rw
	4	*save_manufacturer_parameter* (objects 2000_h ... $5FFF_h$)		unsigned 32	rw

Assignment of the variables

save all parameters

saves the parameters of all objects (if available), which have a read/write (rw) right of access.

save_communication_parameter

saves all communication parameters of those objects (objects 1000_h ... $1FFF_h$, if available), which have a read/write (rw) right of access (here e.g. 1005_h ... 1029_h).

save_application_parameter

saves all application parameters of those objects (objekte 6000_h ... $9FFF_h$, if available), which have a read/write (rw) right of access (here e.g. $6xxx_h$).

save_manufacturer_parameter

saves all manufacturer parameters of those objects (objects 2000_h ... $5FFF_h$, if available), which have a read/write (rw) right of access (here e.g. $2xxx_h$).

The storage mode is shown in the content of this object:
Bit 1 of object 1010_h, sub-index 1 is not set, i.e the CAN-CBX-module does not save the configuration automatically. The storage must be initiated by writing the character string 'save' (73_h 61_h 76_h 65_h, order from CAN telegram) to object 1010_h, sub-index 1-4.

On read access to the appropriate sub-index, the CAN-CBX module provides information about its storage functionality with the format described in the following:

Bit:	31 ... 2	1	0
Inhalt:	reserved	auto	cmd
	0	0	1
	MSB ... LSB		

Bit	Value	Description
auto	0	CAN-CBX module does **not** save the parameters autonomously
	1	CAN-CBX module **saves** the parameters autonomously
cmd	0	CAN-CBX module does **not** save the parameters on command
	1	CAN-CBX module **saves** the parameters on command

Autonomous saving means that the CAN-CBX module stores the storable parameters non-volatile and without a user request.

8.9.13 Restore Default Parameters (1011_h)

INDEX	**1011_h**
Name	*restore default parameters*
Data Type	unsigned 32

Via this command the default parameters, valid when leaving the manufacturer, are restored.
Therefor the parameter groups described below are distinguished.
Every individual setting stored in the EEPROM will be lost.
After a reset the default parameters will be active. The reset of the parameters however must be initiated with a write access to object 1011_h. To write the index a specific signature as shown below has to be transmitted.
Reading the index provides information about its parameter restoring capability (refer to [1] for more information).

Index	Sub-index	Description	Value range	Data type	Access mode
1011_h	0	*number_of_entries*	4	unsigned 8	ro
	1	*restore_all_default_parameters* (objects 1000_h ... $9FFF_h$)	no default, write: 64 61 6F $6C_h$ (= ASCII: ‘d’ ‘a’ ‘o’ ‘l’)	unsigned 32	rw
	2	*restore_communication_parameter* (objects 1000_h ... $1FFF_h$)		unsigned 32	rw
	3	*restore_application_parameter* (objects 6000_h ... $9FFF_h$)		unsigned 32	rw
	4	*restore_manufacturer_parameter* (objects 2000_h ... $5FFF_h$)		unsigned 32	rw

Assignment of the variables

restore all parameters

restores the default parameters of all objects (if available), which have a read/write (rw) right of access.

restore_communication_parameter

restores all communication default parameters of those objects (objects 1000_h ... $1FFF_h$, if available, here e.g. 1005_h ... 1029_h).

restore_application_parameter

restoers all application default parameters of those objects (objekts 6000_h ... $9FFF_h$, if available, here e.g. $6xxx_h$).

restore_manufacturer_parameter

loads all manufacturer default parameters of those objects (objects 2000_h ... $5FFF_h$, if available, here e.g. $2xxx_h$).

Bit 0 of object 1011_h, sub-index 1 is set, i.e. the CAN-CBX module restores the default values initiated by writing the signature 'load' (64_h 61_h $6F_h$ $6C_h$, sequence in CAN telegram) in object 1011_h, sub-index 1-4.

On read access to the appropriate sub-index, the CANopen device provides information about its default parameter restoring capability with the following format:

Bit:	31 ... 1	0
Content:	reserved	cmd
	0	1
	MSB	LSB

Bit	Value	Description
cmd	0	the CAN-CBX-module does **not** restore default parameters
	1	the CAN-CBX-module **restores** the default parameters

8.9.14 COB_ID Emergency Message (1014_h)

INDEX	**1014_h**
Name	*COB-ID emergency object*
Data type	unsigned 32
Default value	80_h + Node-ID

This object defines the COB-ID of the emergency object (EMCY).

The structure of this object is shown in the following table:

Bit-No.	Value	Meaning
31 (MSB)	0/1	0: EMCY exists / is valid 1: EMCY does not exist / EMCY is not valid
30	0	reserved (always 0)
29	0	always 0 (11-bit ID)
28...11	0	always 0 (29-bit IDs are not supported)
10...0 (LSB)	x	bits 0...10 of COB-ID

The identifier can take values between 0...$7FF_h$.

8.9.15 Inhibit Time EMCY (1015_h)

INDEX	**1015_h**
Name	*inhibit_time_emergency*
Data type	unsigned 16
Access mode	rw
Value range	0...$FFFF_h$
Default value	0

The *Inhibit Time* for the EMCY message can be defined with this entry. The time is determined as a multiple of 100 µs.

8.9.16 Consumer Heartbeat Time (1016_h)

INDEX	**1016_h**
Name	*consumer heartbeat time*
Data type	unsigned 32
Default value	No

The heartbeat function can be used for mutual monitoring of the CANopen modules (especially to detect connection failures). Unlike node guarding/life guarding the heartbeat function does not require RTR-Frames.

Function:
A module, the so-called heartbeat producer, cyclically transmits a heartbeat message on the CAN-bus on the node-guarding identifier (see object $100E_h$). One or more heartbeat consumers receive the message. It has to be received within the heartbeat time stored on the heartbeat consumer, otherwise a heartbeat event is triggered on the heartbeat-consumer module. A heartbeat event generates a heartbeat error on the CAN-CBX module.

Each module can act as a heartbeat producer and a heartbeat consumer. The CAN-CBX module can represent at most one heartbeat consumer per CAN net.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1016_h	0	*number_of_entries*	1	1	unsigned 8	ro
	1	*consumer_heartbeat_time*	$0...007FFFFF_h$	0	unsigned 32	rw

Meaning of the variable *consumer-heartbeat_time_x*:

	consumer-heartbeat_time_x		
Bit	31 24	23 16	15 0
Assignment	reserved (always '0')	*Node-ID* (unsigned 8)	*heartbeat_time* (unsigned 16)

Node-ID Node-Id of the heartbeat producer to be monitored.

heartbeat_time Within this time [ms] the heartbeat producer has to transmit the heartbeat on the node-guarding ID, to avoid the transmission of a heartbeat event.
The consumer-heartbeat time of the monitoring module must always be higher than the producer-heartbeat time of the heartbeat-transmitting module.

Example:

consumer-heartbeat_time = $0031\ 03E8_h$

=> *Node-ID* = 31_h = 49_d
=> *heartbeat time* = $3E8_h$ = 1000_d => 1 s

8.9.17 Producer Heartbeat Time (1017_h)

INDEX	**1017_h**
Name	*producer heartbeat time*
Data type	unsigned 16
Default value	0 ms

The producer heartbeat time defines the cycle time with which the CAN-CBX- module transmits a heartbeat-frame to the node-guarding ID.

If the value of the producer heartbeat time is higher than ‘0’, it is active and stops the node-/ life-guarding (see page 64).
If the value of the producer-heartbeat-time is set to ‘0’, transmitting heartbeats by this module is stopped.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1017_h	0	*producer-heartbeat_time*	0...$FFFF_h$	0 ms	unsigned 16	rw

producer-heartbeat_time Cycle time [ms] of heartbeat producer to transmit the heartbeat on the node-guarding ID (see object $100E_h$).
The consumer-heartbeat time of the monitoring module must always be higher than the producer-heartbeat time of the heartbeat-transmitting module.

8.9.18 Identity Object (1018_h)

INDEX	**1018_h**
Name	*identity object*
Data type	unsigned 32
Default value	No

This object contains general information to the CAN module.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1018_h	0	*no_of_entries*	4	4	unsigned 8	ro
	1	*vendor_id*	0...$FFFFFFFF_h$	0000 0017_h	unsigned 32	ro
	2	*product_code*	0...$FFFFFFFF_h$	see chapter 8.9.1 (page 54)	unsigned 32	ro
	3	*revision_number*	0...$FFFFFFFF_h$	0	unsigned 32	ro
	4	*serial_number*	0...$FFFFFFFF_h$	-	unsigned 32	ro

Description of the variables:

vendor_id This variable contains the esd-vendor-ID. This is always 0000 0017_h.

product_code Here the esd-article number of the product is stored.
The nibbles of the long words have the following meaning:

product_code = `abcd efgh`$_h$

`a`: 1... article number beginning with character “K”
2....article number beginning with character “C”

`bcde`: 4-digit hex number, which is interpreted as the integer part of the decimal number (on the left of the decimal point).

`f`: currently not evaluated

`gh`: 2-digit hex number, which is interpreted an fraction part of the decimal number (on the right of the decimal point).

Example: ‘2303 2002_h’ corresponds to article number ‘C.3020.02’.

revision_number Here the software version is stored. In accordance with [1] the two MSB represent the revision numbers of the major changes and the two LSB show the revision number of minor corrections or changes.

revision_no	
major_revision_no 31 16	*minor_revision_no* 15 0
MSB	LSB

serial_number Here the serial number of the hardware is read. The first two characters of the serial number are letters which designate the manufacturing lot. The following characters represent the actual serial number.

In the two MSB of *serial_no* the letters of the manufacturing lot are coded. They each contain the ASCII-code of the letter with the MSB set '1' in order to be able to differentiate between letters and numbers:
(ASCII-Code) + 80_h = read_byte

The two last significant bytes contain the number of the module as BCD-value.

Example:
If the value 'C1C2 0105_h' is being read, this corresponds to the hardware-serial number code 'AB 0105'. This value has to correspond to the serial number of the module.

8.9.19 Synchronous Counter Overflow Value (1019_h)

INDEX	**1019_h**
Name	*Synchronous_Counter_Overflow*
Data type	unsigned 8
Default value	0

This object defines whether a counter is mapped into the SYNC message or not and further the highest value the counter can reach.

The value range of the object is described in the following table:

Value	Description
0	The SYNC message shall be transmitted as a CAN message of data length ‘0’.
1	reserved
2...240	The SYNC message shall be transmitted as a CAN message of data length ‘1’. The first data byte contains the counter.
241...255	reserved

8.9.20 Verify Configuration (1020_h)

INDEX	**1020_h**
Name	*verify configuration*
Data type	unsigned 32
Default value	No

In this object the date and the time of the last configuration can be stored to check later whether the configuration complies with the expected configuration or not.
The content of the parameters is not evaluated by the firmware.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1020_h	0	*no_of_entries*	2	2	unsigned 8	ro
	1	*configuration_date*	0...$FFFFFFFF_h$	0	unsigned 32	rw
	2	*configuration_time*	0...$FFFFFFFF_h$	0	unsigned 32	rw

Parameter Description:

configuration_date Date of the last configuration of the module. The value is defined in number of days since the 01.01.1984.

configuration_time Time in ms since midnight at the day of the last configuration.

8.9.21 Error Behaviour Object (1029_h)

INDEX	1029_h
Name	*error behaviour object*
Data type	unsigned 8
Default value	No

If an error event occurs (such as heartbeat error), the module changes into the status which has been defined in variable *communication_error* or *output_error*.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1029_h	0	*no_of_error_classes*	1	1	unsigned 8	ro
	1	*communication_error*	0...2	0	unsigned 8	rw

Meaning of the variables:

Variable	Meaning
no_of_error_classes	number of error-classes (here always ‘1’)
communication_error	heartbeat/lifeguard error and *Bus off*

The module can enter the following states if an error occurs.

Variable	Module state
0	pre-operational (only if the current state is operational)
1	no state change
2	stopped

8.9.22 NMT Startup ($1F80_h$)

INDEX	**$1F80_h$**
Name	*NMT startup*
Data type	unsigned 32
Default value	2

The NMT startup is implemented to be able to start CANopen nodes in environments without NMT-master.
Via NMT startup the auto startup of a CANopen node can be switched on or off.
Further features of the parameters *NMT startup* are currently not supported.

The value range of the object is described in the following table:

Value	Meaning
$0000\ 0002_h$	Auto startup disabled (default)
$0000\ 0008_h$	Auto startup enabled
all other values	reserved

8.9.23 Self Starting Nodes Timing Parameters ($1F91_h$)

INDEX	**$1F91_h$**
Name	*Self starting nodes timing parameters*
Data type	unsigned 16

Index	Sub-index	Description	Value range	Default	Data type	Access mode
$1F91_h$	0	*number_of_entries*	1	1	unsigned 8	ro
	1	*NMT master detection timeout*	0...$FFFF_h$	64_h	unsigned 16	rw

Sub-index 1 of this object contains the timeout in [ms] between the change from “preoperational” > “operational”. In default it is 100 ms.

The sub-indices 2 and 3 of this object are not supported.

8.9.24 Object Transmit PDO Communication Parameter 1800_h - 1803_h

This objects define the parameters of the transmit-PDOs.

INDEX	**1800_h - 1803_h**
Name	*transmit PDO parameter*
Data Type	PDOCommPar

Index	Sub-index	Description	Value range	Default	Data type	Access
1800_h	0	*number_of_entries*	5	5	unsigned 8	ro
	1	*COB-ID used by PDO*	1...$C00007FF_h$	40000180_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF_h	FF_h	unsigned 8	rw
	3	*inhibit time*	0...$FFFF_h$	0	unsigned 16	rw
	4	*reserved*	0..FF_h	0	unsigned 8	const
	5	*event timer*	0...$FFFF_h$	0	unsigned 16	rw
1801_h	0	*number_of_entries*	5	5	unsigned 8	ro
	1	*COB-ID used by PDO*	1...$C00007FF_h$	40000280_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF_h	FF_h	unsigned 8	rw
	3	*inhibit time*	0...$FFFF_h$	0	unsigned 16	rw
	4	*reserved*	0..FF_h	0	unsigned 8	const
	5	*event timer*	0...$FFFF_h$	0	unsigned 16	rw
1802_h	0	*number_of_entries*	5	5	unsigned 8	ro
	1	*COB-ID used by PDO*	1...$C00007FF_h$	40000380_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF_h	FF_h	unsigned 8	rw
	3	*inhibit time*	0...$FFFF_h$	0	unsigned 16	rw
	4	*reserved*	0..FF_h	0	unsigned 8	const
	5	*event timer*	0...$FFFF_h$	0	unsigned 16	rw
1803_h	0	*number_of_entries*	5	5	unsigned 8	ro
	1	*COB-ID used by PDO*	1...$C00007FF_h$	40000480_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF_h	FF_h	unsigned 8	rw
	3	*inhibit time*	0...$FFFF_h$	0	unsigned 16	rw
	4	*reserved*	0..FF_h	0	unsigned 8	const
	5	*event timer*	0...$FFFF_h$	0	unsigned 16	rw

Value range refer [1]
The *transmission types* 0, 1...240, 254 and 255 (FF_h) are supported.

NOTICE
Always RTR-disabled $40000xxx_h$!

8.9.25 Transmit PDO Mapping Parameter $1A00_h$ - $1A02_h$

This objects define the assignment of the transmit data to the Tx-PDOs.

INDEX	**$1A00_h$ - $1A03_h$**
Name	*transmit PDO mapping*
Data Type	PDO Mapping

The following table shows the assignment of the transmit PDO mapping parameters:

Index	Sub-index	Description	Value range	Default	Data type	Access
	0	*number of entries*	2	**1**	unsigned 8	rw
$1A00_h$	1	*Process_Value_1*	0...$FFFFFFFF_h$	9130 0120_h	unsigned 32	rw
	2	*(Field_Value_1)*	0...$FFFFFFFF_h$	(9100 0120_h)	unsigned 32	rw
	0	*number of entries*	2	**1**	unsigned 8	rw
$1A01_h$	1	*Process_Value_2*	0...$FFFFFFFF_h$	9130 0220_h	unsigned 32	rw
	2	*(Field_Value_2)*	0...$FFFFFFFF_h$	(9100 0220_h)	unsigned 32	rw
	0	*number of entries*	2	**1**	unsigned 8	rw
$1A02_h$	1	*Process_Value_3*	0...$FFFFFFFF_h$	9130 0320_h	unsigned 32	rw
	2	*(Field_Value_3)*	0...$FFFFFFFF_h$	(9100 0320_h)	unsigned 32	rw
	0	*number of entries*	2	**1**	unsigned 8	rw
$1A03_h$	1	*Process_Value_4*	0...$FFFFFFFF_h$	9130 0420_h	unsigned 32	rw
	2	*(Field_Value_4)*	0...$FFFFFFFF_h$	(9100 0420_h)	unsigned 32	rw

Value range refer [1], table 10, 11.

Only the objects listed in the column “Default” can be mapped.

Structure of the PDO mapping using the example of object $1A00_h$, sub-index 01_h:

31 ... 16	15 ... 8	7 ... 0
Index e.g.: 9130_h	Subindex e.g.: 01_h	Length e.g.: 20_h
MSB		LSB

8.10 Device Profile Area

8.10.1 Overview of the Implemented Objects 6110_h ...9135_h

Index	Name	Data Type
6110_h	*AI_sensor_type*	unsigned 16
6111_h	*AI_autocalibration*	unsigned 32
6112_h	*AI_operating_mode*	unsigned 8
6114_h	*AI_ADC_sampling_rate*	unsigned 32
6131_h	*AI_physical_unit_PV*	unsigned 32
6132_h	*AI_decimal_digits_PV*	unsigned 8
6150_h	*AI_status*	unsigned 8
9100_h	*AI_FV*	integer 32
9103_h	*AI_interrupt_delta_input_FV*	integer 32
9130_h	*AI_PV*	integer 32
9133_h	*AI_interrupt_delta_input_PV*	integer 32
9134_h	*AI_interrupt_lower_limit_PV*	integer 32
9135_h	*AI_interrupt_upper_limit_PV*	integer 32

An overview of the cooperation of the objects of the “Device Profile Area” ($61xx_h$ and $91xx_h$) and the “Manufacturer Specific Profile Area” (24xxh) is shown in the diagram on the following page.

8.10.2 Relationship Between the Implemented Analog Input Objects

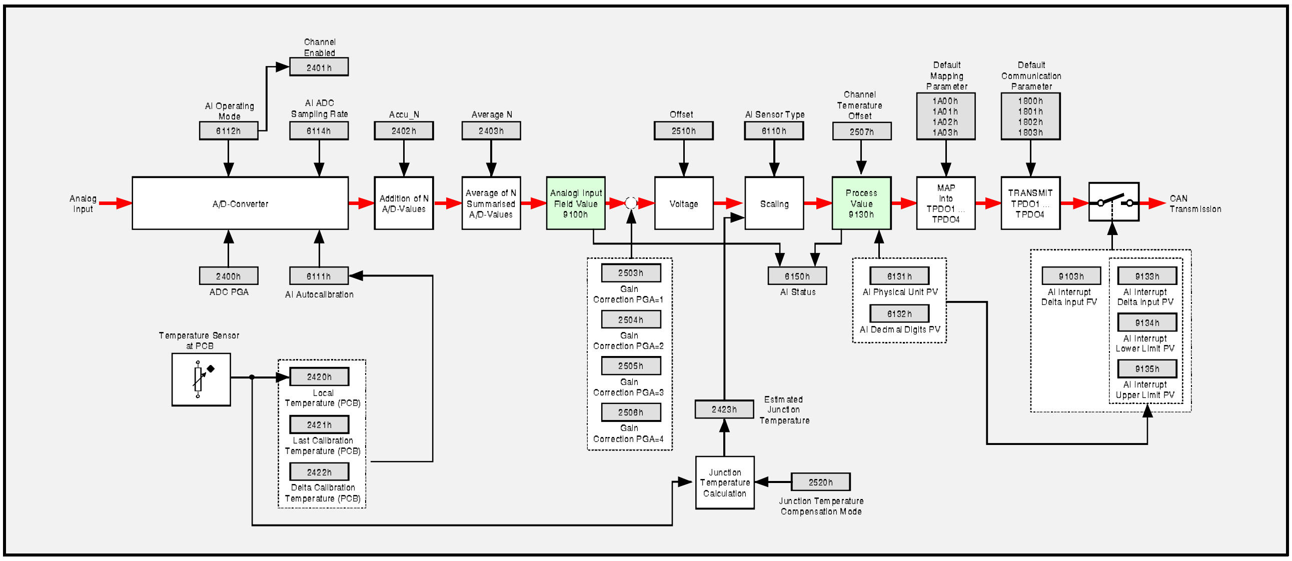


Fig. 21: Relationship between the implemented objects

8.10.3 AI Sensor Type (6110_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
6110_h	0	*number_of_entries*	4	4	unsigned 8	ro
	1	*AI_sensor_type_1*	1 ...9, 28_h	28_h	unsigned 16	rw
	2	*AI_sensor_type_2*	1 ...9, 28_h	28_h	unsigned 16	rw
	3	*AI_sensor_type_3*	1 ...9, 28_h	28_h	unsigned 16	rw
	4	*AI_sensor_type_4*	1 ...9, 28_h	28_h	unsigned 16	rw

Description of the parameter *AI_sensor_type_x* (x = 1...4):

The parameter contains the type of the connected thermocouple. The end number of the variable name is the number of the thermocouple interface (see also [4]).

Value range:

Value of the parameter	Description
1	thermocouple type J
2	thermocouple type K
3	thermocouple type L
4	thermocouple type N
5	thermocouple type R
6	thermocouple type S
7	thermocouple type T
8	thermocouple type B
9	thermocouple type U
28_h	voltage measurement

INFORMATION
Please note, when connecting thermocouples, the type of the connected thermocouple has to be set here.
The default value of the parameter *AI_sensor_type_x* is 40 (28_h), which is the value for the voltage measurement.

8.10.4 AI Autocalibration (6111_h)

<table>
<tr><th>Index</th><th>Sub-index</th><th>Description</th><th>Value range</th><th>Default</th><th>Data type</th><th>Access</th></tr>
<tr><td rowspan="5">6111_h</td><td>0</td><td>number_of_entries</td><td>4</td><td>4</td><td>unsigned 8</td><td>ro</td></tr>
<tr><td>1</td><td>AI_autocalibration_1</td><td colspan="2" rowspan="4">no default, write:
69 6C 61 63_h
(= ASCII: ‘i’ ‘l’ ‘a’ ‘c’)</td><td>unsigned 32</td><td>wo</td></tr>
<tr><td>2</td><td>AI_autocalibration_2</td><td>unsigned 32</td><td>wo</td></tr>
<tr><td>3</td><td>AI_autocalibration_3</td><td>unsigned 32</td><td>wo</td></tr>
<tr><td>4</td><td>AI_autocalibration_4</td><td>unsigned 32</td><td>wo</td></tr>
</table>

Function:
Writing the ASCII-strings “cali” leads to the automatic-calibration of the corresponding channel.

The automatic-calibration starts automatically:

- after detection of new thermocouples
- after change of the parameter
  - Sampling-Rate *AI_ADC_sampling_rate* (object 6114_h) or
  - Gain *ADC_PGA* (object 2400_h)
- after exceeding a defined change in temperature of the circuit board since the last calibration (if enabled)
  - temperature measured during the last calibration
    last_calibration_temp (object 2421_h)
  - maximum allowable temperature difference since the last calibration
    delta_calibration_temp (object 2422_h)

8.10.5 AI Operating Mode (6112_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
6112	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_operating_mode_1*	0, 1	1	unsigned 8	rw
	2	*AI_operating_mode_2*	0, 1	1	unsigned 8	rw
	3	*AI_operating_mode_3*	0, 1	1	unsigned 8	rw
	4	*AI_operating_mode_4*	0, 1	1	unsigned 8	rw

Description of parameter *AI_operating_mode_x* (x = 1...4):

The parameter choses the desired operation mode. It contains the “nominal-value” of the operating mode. The “actual value” is readable via object 2401_h.

Value range:

Value of the parameter	Operation mode of the channel
00_h	channel disabled
01_h	normal operation

8.10.6 AI ADC Sampling Rate (6114_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
6114_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_ADC_sampling_rate_1*	$03E8_h$...$61A80_h$	$61A80_h$	unsigned 32	rw
	2	*AI_ADC_sampling_rate_2*	$03E8_h$...$61A80_h$	$61A80_h$	unsigned 32	rw
	3	*AI_ADC_sampling_rate_3*	$03E8_h$...$61A80_h$	$61A80_h$	unsigned 32	rw
	4	*AI_ADC_sampling_rate_4*	$03E8_h$...$61A80_h$	$61A80_h$	unsigned 32	rw

Description of the parameter *AI_ADC_sampling_rate_x* (x = 1...4):

Via this parameter the sampling rates of the single channels can be set each independent from the others. The resolution is 1 µs. The following values are supported:

Value range:

Allowed parameter values *2)	Sample time	Sampling rate
1 000 ($03E8_h$)	1 ms	1000 Hz
2 000	2 ms	500 Hz
10 000	10 ms	100 Hz
16 667	16.667 ms	60 Hz
20 000	20 ms	50 Hz
33 333	33.333 ms	30 Hz
40 000	40 ms	25 Hz
66 667	66.667 ms	15Hz
100 000	100 ms	10 Hz
200 000	200 ms	5 Hz
400 000 ($61A80_h$)	400 ms	2.5 Hz

*2) If a value is transmitted which is not in the list of allowed parameters, it will be rounded to the next allowed value.

INFORMATION
To achieve a high resolution, it is advisable to use sample-times, which are a multiple of 20 ms (50 Hz), because that will reduce disturbances caused by the power frequency!

8.10.7 AI Physical Unit (6131_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
6131_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_physical_unit_1*	002D 0000_h, FA26 0000_h	$unit_{SensorType}$_1	unsigned 32	ro
	2	*AI_physical_unit_2*	002D 0000_h, FA26 0000_h	$unit_{SensorType}$_2	unsigned 32	ro
	3	*AI_physical_unit_3*	002D 0000_h, FA26 0000_h	$unit_{SensorType}$_3	unsigned 32	ro
	4	*AI_physical_unit_4*	002D 0000_h, FA26 0000_h	$unit_{SensorType}$_4	unsigned 32	ro

The default values $unit_{SensorType}$_x (x = 1 - 4) depend on the sensor-type set.: With the default-setting of object 6110_h =28_h (40 = voltage measurement) it would be FA26 0000_d = voltage in µV.

Description of the variable *AI_physical_unit_x* (x = 1...4):

These variables contain the physical units and prefixes for the measuring channels used. The value of the variables is determined by the sensor type specified in object "*AI_sensor_type*" (object 6110_h).

The structure of the variable is defined in CiA 404. The coding of the physical units and prefixes is done according to CiA303-2.

The structure of the variables is described on the following page.

Structure of the variable:

31 ... 24	23 ... 16	15 ... 8	7 ... 0
Prefix	SI Numerator	SI Denominator	reserved
MSB			LSB

Coding of the prefix for physical units:

Prefix	Factor	Notation index
-	10^0	00_h
micro	10^{-6}	FA_h

SI Numerator and SI Denominator contain the physical units.

Coding of the physical units :

International symbol	Name of unit	Notation index
V	volt	26_h
˚C	degree Celsius	$2D_h$
dimensionless	none	0

Value range:

Sensor type	Physical unit
thermocouple	degree Celsius (Variable value = $002D\ 0000_h$)
voltage measurement	Volt (Variable value = $FA26\ 0000_h$)

8.10.8 AI Decimal Digits PV (6132_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
6132_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_decimal_digits_PV_1*	0...3	3	unsigned 8	rw
	2	*AI_decimal_digits_PV_2*	0...3	3	unsigned 8	rw
	3	*AI_decimal_digits_PV_3*	0...3	3	unsigned 8	rw
	4	*AI_decimal_digits_PV_4*	0...3	3	unsigned 8	rw

Description of the parameter *AI_decimal_digits_PV_x* (x = 1...4):

This parameter contains the number of fractional digits of the process value (PV) in object "*AI_PV*" (9130_h).
The value of the parameter gives the number of the decimal digits.

Example:

AI_decimal_digits_PV_1 = 3
Value in object *AI_PV_1* = 123456_d

Measured value: 123.456 µV (at default setting (voltage measurement) see object 6110_h).

8.10.9 AI Status (6150$_h$)

Index	Sub-index	Description	Value range	Default	Data type	Access
6150$_h$	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_status_1*	0...5	-	unsigned 8	ro
	2	*AI_status_2*	0...5	-	unsigned 8	ro
	3	*AI_status_3*	0...5	-	unsigned 8	ro
	4	*AI_status_4*	0...5	-	unsigned 8	ro

Description of the variable *AI_status_x* (x = 1...4):

These variables return the state of the corresponding analog input channel.

Bit:	7	6	5	4	3	2	1	0
Meaning:	reserved *3)	not supported *3)			reserved *3)	Negative Overload	Positive Overload	Not Valid

*3) These bits are always returned as ‘0’.

Value range:

Description	Negative overload limit	Positive overload limit
thermocouple type J	-210 ˚C	1212 ˚C
thermocouple type K	-200 ˚C	1400 ˚C
thermocouple type L	-200 ˚C	908 ˚C
thermocouple type N	-250 ˚C	1315 ˚C
thermocouple type R	-50 ˚C	1782 ˚C
thermocouple type S	-50 ˚C	1792 ˚C
thermocouple type T	-250 ˚C	405 ˚C
thermocouple type B	50 ˚C	1835 ˚C
thermocouple type U	-200 ˚C	605 ˚C
voltage measurement at default setting	- 0.512 V	+ 0.512 V

A bit is active (set to ‘1’), if the lower or the upper limit is exceeded. The bits 1 and 2 cannot be set at the same time.

If no sensor is connected, only bit “Not Valid” is set.
If the temperature exceeds the range, bits “Not Valid” AND the corresponding “Overload”-bit are set.

Valid return messages are e.g.:

AI_Status_1 = 00_h : no error
AI_Status_1 = 03_h : positive overload and sensor value not valid

8.10.10 Analog Input Field Value (9100_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
9100_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_FV_1*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	2	*AI_FV_2*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	3	*AI_FV_3*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	4	*AI_FV_4*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro

Description of the variable *AI_FV_x* (x = 1...4):

These variables contain the uncorrected “Raw values” of the A/D-converter.

Value range:

AI_FV_x = $8000\ 0000_h$ ... $7FFF\ FFFF_h$

8.10.11 AI Interrupt Delta Input FV (9103_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
9103_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_interrupt_delta_input_FV_1*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw
	2	*AI_interrupt_delta_input_FV_2*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw
	3	*AI_interrupt_delta_input_FV_3*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw
	4	*AI_interrupt_delta_input_FV_4*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw

Description of the parameter *AI_interrupt_delta_input_FV_x* (x = 1...4):

If a field-value is mapped on a PDO and the deviation of the field value is higher than specified in *AI_interrupt_delta_input_FV_*x, this PDO is transmitted.

Value range:

*AI_interrupt_delta_input_FV_*x	Meaning
8000 0000_h ... FFFF $FFFF_h$	for negative values of the parameter a PDO-transmission is initiated with every change of the field value.
0	no comparison of the field values and thus no transmission
0000 0001_h .... 7FFF $FFFF_h$	a PDO-transmission will only be initiated, if the deviation of the field value is higher than the value specified here.

8.10.12 Analog Input Process Value (9130$_h$)

INFORMATION
This object contains the important data (voltage or temperature values)!

Index	Sub-index	Description	Value range	Default	Data type	Access
9130$_h$	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_PV_1*	80000000$_h$ ... 7FFFFFFF$_h$	-	integer 32	ro
	2	*AI_PV_2*	80000000$_h$ ... 7FFFFFFF$_h$	-	integer 32	ro
	3	*AI_PV_3*	80000000$_h$ ... 7FFFFFFF$_h$	-	integer 32	ro
	4	*AI_PV_4*	80000000$_h$ ... 7FFFFFFF$_h$	-	integer 32	ro

Description of the variable *AI_PV_x* (x = 1...4):

This variables return the corrected, measured values of the four channels. The physical units of the values are defined in "*AI_physical_unit_x*" (object 6131$_{h,}$, see page 90).

The number of decimal places of the variable is defined in object "*AI_decimal_digits_PV_x*" (object 6132$_{h,}$ see page 92).

Value range:

AI_PV_x = 8000 0000$_h$ ... 7FFF FFFF$_h$

8.10.13 AI Interrupt Delta Input PV (9133_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
9133_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_interrupt_delta_input_PV_1*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw
	2	*AI_interrupt_delta_input_PV_2*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw
	3	*AI_interrupt_delta_input_PV_3*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw
	4	*AI_interrupt_delta_input_PV_4*	80000000_h ... $7FFFFFFF_h$	0	integer 32	rw

Description of the parameter *AI_interrupt_delta_input_PV_x* (x = 1...4):

If a process variable is mapped on a PDO and the deviation of the process variable is higher than specified in *AI_interrupt_delta_input_PV*_x, this PDO is transmitted.

The number of decimal places of the variable is defined in object "*AI_decimal_digits_PV_x*" (object $6132_{h,}$ see page 92).

Value range:

*AI_interrupt_delta_input_PV*_x	Meaning
8000 0000_h ... FFFF $FFFF_h$	for negative values of the parameter a PDO-transmission is initiated with every change of the process values
0	no comparison of the process values and thus no transmission
0000 0001_h .... 7FFF $FFFF_h$	a PDO-transmission will only be initiated, if the deviation of the process values is higher than the value specified here.

8.10.14 AI Interrupt Lower Limit PV (9134_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
9134_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_interrupt_lower_limit_PV_1*	80000000_h ... $7FFFFFFF_h$	80000000_h	integer 32	rw
	2	*AI_interrupt_lower_limit_PV_2*	80000000_h ... $7FFFFFFF_h$	80000000_h	integer 32	rw
	3	*AI_interrupt_lower_limit_PV_3*	80000000_h ... $7FFFFFFF_h$	80000000_h	integer 32	rw
	4	*AI_interrupt_lower_limit_PV_4*	80000000_h ... $7FFFFFFF_h$	80000000_h	integer 32	rw

Description of the parameter *AI_interrupt_lower_limit_PV_x* (x = 1...4):

If a process variable is mapped on a PDO and the deviation of the process variable is lower than specified in *AI_interrupt_lower_limit_PV_*x, this PDO is transmitted.

The number of decimal places of the variable is defined in object "*AI_decimal_digits_PV_x*" (object $6132_{h,}$ see page 92).

Value range:

*AI_interrupt_lower_limit_PV_*x = 8000 0000_h ... 7FFF $FFFF_h$

8.10.15 AI Interrupt Upper Limit PV (9135$_h$)

Index	Sub-index	Description	Value range	Default	Data type	Access
9135$_h$	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*AI_interrupt_upper_limit_PV_1*	80000000$_h$ ... 7FFFFFFF$_h$	7FFFFFFF$_h$	integer 32	rw
	2	*AI_interrupt_upper_limit_PV_2*	80000000$_h$ ... 7FFFFFFF$_h$	7FFFFFFF$_h$	integer 32	rw
	3	*AI_interrupt_upper_limit_PV_3*	80000000$_h$ ... 7FFFFFFF$_h$	7FFFFFFF$_h$	integer 32	rw
	4	*AI_interrupt_upper_limit_PV_4*	80000000$_h$ ... 7FFFFFFF$_h$	7FFFFFFF$_h$	integer 32	rw

Description of the parameter *AI_interrupt_upper_limit_PV_x* (x = 1...4):

If a process variable is mapped on a PDO and the deviation of the process variable is higher than specified in *AI_interrupt_upper_limit_PV_*x, this PDO is transmitted.

The number of decimal places of the variable is defined in object "*AI_decimal_digits_PV_x*" (object 6132$_{h,}$ see page 92).

Value range:

*AI_interrupt_upper_limit_PV_*x = 8000 0000$_h$ ... 7FFF FFFF$_h$

8.11 Manufacturer Specific Profile Area

8.11.1 Overview of Manufacturer Specifc Objects (2400_h ... 2510_h)

Index	Name	Data Type
2400_h	*ADC_PGA*	unsigned 8
2401_h	*Channel_enabled*	unsigned 8
2402_h	*Accu_N*	unsigned 8
2403_h	*Average_N*	unsigned 8
2420_h	*Local_temperature (PCB)*	integer 16
2421_h	*Last_calibration_temperature (PCB)*	integer 16
2422_h	*Delta_calibration_temperature (PCB)*	integer 16
2423_h	*Estimated_junction_temperature* *4)	integer 16
2503_h	*Gain_correction*	integer 16
2504_h	*Gain_correction_PGA=2*	integer 16
2505_h	*Gain_correction_PGA=3*	integer 16
2506_h	*Gain_correction_PGA=4*	integer 16
2507_h	*Channel_temperature_offset*	integer 16
2510_h	*U_Offset*	integer 16
2520_h	*Junction_Temperature_Slope_Compensation_Mode* *5)	integer 16

*4) This object is only supported from firmware version V2.01 on.

*5) This object is not supported yet.

See also diagram “Relationship Between the Implemented Objects for the Analog Inputs” on page ?.

8.11.2 ADC_PGA (2400_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2400_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*ADC_PGA_1*	1...4	3	unsigned 8	rw
	2	*ADC_PGA_2*	1...4	3	unsigned 8	rw
	3	*ADC_PGA_3*	1...4	3	unsigned 8	rw
	4	*ADC_PGA_4*	1...4	3	unsigned 8	rw

Description of the parameter *ADC_PGA_x* (x = 1...4):

This parameter sets the gain (PGA) of the A/D-converters (ADS1255).

Value range:
The gain V_x is:

$$V_x = 2^{ADC\_PGA\_x}$$

The gain 2, 4, 8 and 16 can be set. The gain is transparent for the user and already included in the calculation of the firmware.

NOTICE
For the connection of thermocouples the following voltage limits have to be considered.

Depending on *ADC_PGA_x* the maximum permissible voltage values are:

Gain (ADC_PGA_x)	Input voltage U_{FS} [V]
1	± 1.024
2	± 1.024
3	± 0.512
4	± 0.256

INFORMATION
The recommended gain is *ADC_PGA_x* =3.

8.11.3 Channel Enabled (2401_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2401_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*channel_enabled_1*	0, 1	-	boolean	ro
	2	*channel_enabled_2*	0, 1	-	boolean	ro
	3	*channel_enabled_3*	0, 1	-	boolean	ro
	4	*channel_enabled_4*	0, 1	-	boolean	ro

Description of the variable *channel_enabled_x* (x = 1...4):

If a sensor has been detected for the measuring channel this variable returns the "actual value"of the object "*AI_Operating_mode_x*" (object 6112_h). It indicates whether the corresponding channel is active.

With object 6150_h "*AI_status_x*" it can be determined whether a sensor has been detected.

Value range:

*channel_enabled*_x	Binary value	Meaning
false	0	A/D-converter channel is off (*AI_operation_mode_x* = 0)
true	1	A/D-converter channel is on (*AI_operation_mode_x* = 1)

8.11.4 Accu N (2402_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2402_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*accu_count_1*	0...8	0	unsigned 8	rw
	2	*accu_count_2*	0...8	0	unsigned 8	rw
	3	*accu_count_3*	0...8	0	unsigned 8	rw
	4	*accu_count_4*	0...8	0	unsigned 8	rw

Description of the parameter *accu_count_x* (x = 1...4):

This parameter defines the number of analog values to be added. The parameter accu_count reduces the data rate while improving the resolution.
The number of accumulations is:

n ... Number of accumulations

$$n = 2^{(accu\_count\_x)}$$

Up to 256 values can be summed up.

Features:
- Filter with decimation
- Improvement of the resolution
- Reduction of the data rate

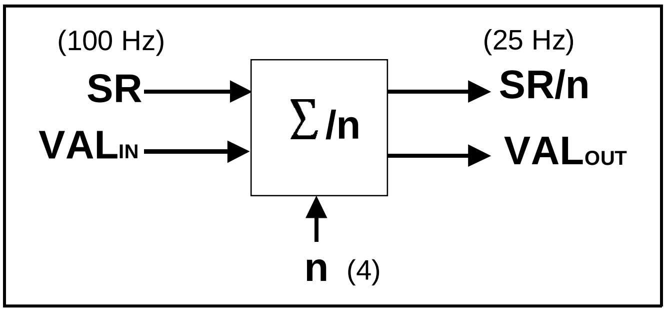
The data rate of the analog value is calculated as:

$$T_{Data} = 2^n \times T\_sampling\_ADC \quad \text{(object } 6114_h\text{)}$$

Description of the Filter:

SR ... sample rate (e.g.:100 Hz)
n ... number of additions (e.g.:4)
SR/n... sample rate/ number of summations (e.g.: 100 Hz / 4 = 25 Hz)

$$Val_{OUT}(t) = \frac{1}{n}\sum_{x=1}^{n} Val_{IN}(t - n + x)$$



8.11.5 Average N (2403_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2403	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*average_count_1*	0...4	0	unsigned 8	rw
	2	*average_count_2*	0...4	0	unsigned 8	rw
	3	*average_count_3*	0...4	0	unsigned 8	rw
	4	*average_count_4*	0...4	0	unsigned 8	rw

Description of the parameter *average_count_x* (x = 1...4):

This parameter defines how many analog values are used to calculate the moving average.
After every conversion a new average is available, because the A/D-values are buffered in a ring buffer.

The number of averaged values is:

m ... Number of averaged values

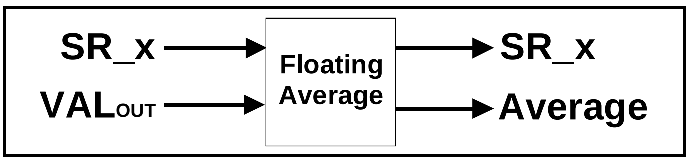
$$m = 2^{(average\_count\_x)}$$

Thus it can be averaged over the last 1, 2, 4, 8 or 16 values.

Features:
- Filter with decimation
- Improvement of the resolution
- A new average is available after every conversion
- Step response is $2^m \cdot T_{Data}$ (see object 2402_h)

INFORMATION
The input for “Average” (object 2403_h) is “Val_{OUT}” configured by object 2402_h!

Description of the Filter:
SR_x ... sample rate



8.11.6 Local Temperature at PCB (2420_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2420_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*local_temperature*	8000_h...$7FFF_h$	-	integer 16	ro

Description of the variable *local_temperature*:

This variable contains the value of the temperature on the circuit board in steps of 1/256 ˚C and is further used for cold-junction compensation (CJC).

The value is coded as described in the following:

$$Temperature[^\circ C] = \frac{Int16\_Value}{256_d}$$

Example:

local_temperature =2640_h => Temperature = 38.25 ˚C

8.11.7 Local Temperature at the Last Calibration (2421_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2421_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*last_calibration_temp_1*	8000_h ... $7FFF_h$	$temp_{\_last\_cal\_}1$	integer 16	ro
	2	*last_calibration_temp_2*	8000_h ... $7FFF_h$	$temp_{\_last\_cal\_}2$	integer 16	ro
	3	*last_calibration_temp_3*	8000_h ... $7FFF_h$	$temp_{\_last\_cal\_}3$	integer 16	ro
	4	*last_calibration_temp_4*	8000_h ... $7FFF_h$	$temp_{\_last\_cal\_}4$	integer 16	ro

The default value $temp_{\_last\_cal\_}x$ (x = 1 - 4) contains the temperature value which is measured at the last calibration.

Description of the variable *last_calibration_temp_x*:

This variable contains the value of the temperature on the circuit board in steps of 1/256 °C, determined at the last calibration of the corresponding channel.

The value is coded as described in the following:

$$Temperature[°C] = \frac{Int16\_Value}{256_d}$$

8.11.8 Calibration Delta Temperature (2422_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2422_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*delta_calibration_temp_1*	0...$FFFF_h$	0	unsigned 16	rw
	2	*delta_calibration_temp_2*	0...$FFFF_h$	0	unsigned 16	rw
	3	*delta_calibration_temp_3*	0...$FFFF_h$	0	unsigned 16	rw
	4	*delta_calibration_temp_4*	0...$FFFF_h$	0	unsigned 16	rw

Description of the variable *delta_calibration_temp_x*:

The ADS1255 performs a self-calibration, if the temperature of the PCB (2420_h) deviates more than the value specified in *delta_calibration_temp_x* from the *last_calib_temp_x* .

Value *delta_calibration_temp_x*	Function
0	no automatic self-calibration at deviation of the temperature
0001_h...$FFFF_h$	automatic self-calibration at exceeding the allowed deviation of the temperature

The value is coded as:

$$Temperature[°\,C] = \frac{Int16\_Value}{256_d}$$

Example:

delta_calibration_temp_1 = 0180_h => Temperature deviation = 1.50 °C

8.11.9 Estimated Junction Temperature (2423_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2423_h *4)	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*estimated_junction_temp_1*	8000_h...$7FFF_h$	-	integer 16	ro
	2	*estimated_junction_temp_2*	8000_h...$7FFF_h$	-	integer 16	ro
	3	*estimated_junction_temp_3*	8000_h...$7FFF_h$	-	integer 16	ro
	4	*estimated_junction_temp_4*	8000_h...$7FFF_h$	-	integer 16	ro

*4) This object is only supported from firmware version V2.01 on.

Description of the variable *estimated_junction_temp_x*:

This variable contains the estimated values of the channel-dependent junction temperature on the circuit board in steps of 1/256 °C.

The value is coded as described in the following:

$$Temperature[^\circ C] = \frac{Int16\_Value}{256_d}$$

Calculation of the *estimated_junction_temperature*

estimated_junction_temp = local_temp + channel_temp_offset + slope_compensation_mode

with

local_temp [Object 2420_h]: value of the temperature on the circuit board

channel_temp_offset
[Object 2507_h]: temperature offset value for the compensation of the temperature drop between the local PCB temperature and the real connector temperature.

slope_compensation_mode
[Object 2520_h]: enables or disables a junction temperature slope compensation. Current *slope_compensation_mode* is = ‘0'.

8.11.10 Calibration Data

The analog inputs of the CAN-CBX-Thermo module have been calibrated by the manufacturer before delivery. The following calibrations have been made:

- ADC gain PGA=1
- ADC gain PGA=2
- ADC gain PGA=3
- ADC gain PGA=4

With command *Restore Default Parameters* (1011_h) the initial setting of the calibration data adjusted by esd are reactivated.

8.11.10.1 Calibration and Process/Field-Value Calculation

In the following formulas the parameter names are shown without the channel identifier at the end of the parameter names to improve the clarity.

The calculations shown in the following are automatically done by the local firmware. For the user the corrected measuring values are accessible by the PV.

Field-Value Calculation

$$AI\_FV = \frac{adc\_raw\_value}{2^{ADC\_PGA}}$$

with

`adc_raw value`:	A/D-converter value after addition on averaging
`AI_FV` [Object 9100_h]:	Field Value
`ADC_PGA` [Object 2400_h]:	ADC-gain

Process-Value Calculation

1) Correction FV

$$FV_{corr} = AI\_FV\left(1 + \frac{corr}{2^{22}}\right)$$

with

FV_{corr}:	internal corrected field value in ADC-units
`AI_FV` [Object 9100_h]:	analog input field value
`corr`:	internal correction factor

$$corr = gc\_PGA1[+ gc\_PGAy]$$

with

`gc_PGA1` [Object 2503_h]:	gain correction factor PGA=1
`gc_PGAy` [Objects 2504_h, 2505_h, 2506_h]:	additional gain correction for the gain (*y*= 2,3,4) PGA=2 [2504_h], PGA=3 [2505_h] und PGA=4 [2506_h]

2) Calculation PV_U (sensor voltage)

$$PV_U = (k \times FV_{corr}) + u\_offset$$

with

PV_U :	process value, voltage measurement, unit [nV]
k: . . .	internal conversion factor, always 1.9073 nV
`u_offset` [Object 2510_h]:	voltage offset value for the compensation of the systematic measuring error

3) Temperature Calculation and CJC

$$PV_T = Tab\left(PV_U\right)$$

with

PV_T [Object 9130_h]: process value (mapped to PDOs)
PV_T = f (SensorType [obj. 6110_h] , PV_U, EJT [object 2423_h])

`Tab`: application of the sensor-type conversion table
Tab = f (*AI_sensor_type* [object 6110_h])

8.11.10.2 Disable Cold Junction Compensation (2425_h)

i	**INFORMATION** Object 2425_h is only intended for test and calibration purposes!

Index	Sub-index	Description	Value range	Default	Data type	Access
2425_h	0	*Ignore_ADT7320*	0,1	0	unsigned 8	rw

Description of the parameter *Ignore_ADT7320*:

By means of this parameter the automatic cold junction compensation can be disabled. Only if this parameter is disabled the estimated junction temperature can be set manually.

1. If the value is set to ‘1’, the variable “*estimated_junction_temp_X*” (index 2423_h , sub-index 1...4) is no longer supplied by the firmware. The previous value is kept.

2. Now the variables “*estimated_junction_temp_X*” (index 2423_h , sub-index 1...4) can be set manually. If they are set for example to the value ‘0’, than there will be no compensation.

Value range:

Ignore_ADT7320	Meaning
0	Automatic cold junction compensation is enabled (object 6150_h can **not** be written)
1	Automatic cold junction compensation is disabled (object 6150_h, sub-index 1..4 can be written)

8.11.10.3 Gain Correction PGA=1/2/3/4 (2503_h - 2506_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*gain_correction_PGA=1_1*	8000_h...$7FFF_h$	*[6]	integer 16	rw
2503_h	2	*gain_correction_PGA=1_2*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	3	*gain_correction_PGA=1_3*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	4	*gain_correction_PGA=1_4*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*gain_correction_PGA=2_1*	8000_h...$7FFF_h$	*[6]	integer 16	rw
2504_h	2	*gain_correction_PGA=2_2*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	3	*gain_correction_PGA=2_3*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	4	*gain_correction_PGA=2_4*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*gain_correction_PGA=3_1*	8000_h...$7FFF_h$	*[6]	integer 16	rw
2505_h	2	*gain_correction_PGA=3_2*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	3	*gain_correction_PGA=3_3*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	4	*gain_correction_PGA=3_4*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*gain_correction_PGA=4_1*	8000_h...$7FFF_h$	*[6]	integer 16	rw
2506_h	2	*gain_correction_PGA=4_2*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	3	*gain_correction_PGA=4_3*	8000_h...$7FFF_h$	*[6]	integer 16	rw
	4	*gain_correction_PGA=4_4*	8000_h...$7FFF_h$	*[6]	integer 16	rw

*[6] The default values have been determined individually for every module at the calibration. Thus these values are not described in this general documentation.

Description of the parameter *gain_correction_PGA=y_x* (y = 1, 2, 3, 4 ; x = 1...4):

These objects contain the gain-correction factors for the PGA-values 1, 2, 3 and 4, as determined during calibration.

Value range: 8000_h ... $7FFF_h$

8.11.11 Channel Temperature Offset (2507_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2507_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*channel_temp_offset_1*	8000_h...$7FFF_h$	0	integer 16	rw
	2	*channel_temp_offset_2*	8000_h...$7FFF_h$	0	integer 16	rw
	3	*channel_temp_offset_3*	8000_h...$7FFF_h$	0	integer 16	rw
	4	*channel_temp_offset_4*	8000_h...$7FFF_h$	0	integer 16	rw

Description of the variable *channel_temp_offset_x* (x = 1...4)

The variable *channel_temp_offset_x* contains the temperature offset of the cannel x in m˚C.

It is used for the compensation of the temperature drop between local PCB-temperature (*local_ temperature*, 2420_h) and real connector temperature.

The corrected local temperature value of channel x is calculated as:

local_temperature_corr_x = *local_temperature* - *channel_temp_offset_x*
(+)

Example:

channel_temp_offset_1 = 1234_d => 1234 m˚C => 1.234 ˚C

8.11.11.1 U_Offset (2510_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2510_h	0	*Number_of_entries*	4	4	unsigned 8	ro
	1	*u_offset_1*	8000_h...$7FFF_h$	0	integer 16	rw
	2	*u_offset_2*	8000_h...$7FFF_h$	0	integer 16	rw
	3	*u_offset_3*	8000_h...$7FFF_h$	0	integer 16	rw
	4	*u_offset_4*	8000_h...$7FFF_h$	0	integer 16	rw

Description of the parameter *u_offset_x* (x = 1...4):

By means of this objects the measurement offset voltage can be compensated. It contains the voltage offset in steps of nV.

If the compensation of an error is necessary, the negative value of the error has to be entered.

Example:

U_{Line_x} = 1.234 µV => 1234_d => *u_offset_x* = -1234_d

8.11.11.2 Junction Temperature Slope Compensation Mode (2520_h)

Index	Sub-index	Description	Value range	Default	Data type	Access
2520_h *5)	0	*slope_compensation_mode*	0	0	unsigned 8	ro

*5) This object is not supported yet.

Description of the parameter *slope_compensation_mode:*

With the *slope_compensation_mode* a junction temperature slope compensation can be enabled or disabled for all channels.
Current *slope_compensation_mode* is always = ‘0', i.e. junction temperature slope compensation is disabled, because the junction temperature compensation is not yet implemented.

Value *slope_compensation_mode*	Function
0	junction temperature slope compensation disabled

8.12 Firmware Management via DS 302-Objects ($1F50_h$...$1F52_h$)

The objects described below are used for program updates via the object dictionary.

NOTICE
The firmware update must be carried out only by qualified personnel!
Faulty program update can result in deleting of the memory and loss of the firmware.
The module then can not be operated further!

INFORMATION
esd offers the program CANfirmdown for a firmware update.
Please contact our support for this.

In normal CiA 301 mode the object $1F50_h$ can not be accessed.
The objects $1F51_h$ and $1F52_h$ are also available in normal CiA 301mode.
For further information about the objects and the firmware-update please refer to [2].

Index	Sub-index	Description	Data type	Access mode
$1F50_h$	0	Boot-Loader: Firmware download	domain	rw
$1F51_h$	1	Boot-Loader: FLASH command	unsigned 8	rw
$1F52_h$	0,1,2	Boot-Loader: Firmware date	unsigned 32	ro

8.12.1 Download Control via Object ($1F51_h$)

INDEX	**$1F51_h$**
Name	Program Control
Data type	unsigned 8
Access type	rw
Value range	0...FE_h
Default value	0

INFORMATION
The value range of this objects in the implementing of the CAN-CBX-module differs from the value range specified in [2].
For further information about object $1F51_h$ and the firmware-update please refer to [2]

8.12.2 Verify Application Software ($1F52_h$)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
$1F52_h$	0	*Number of entries*	2	2	unsigned 8	ro
	1	*Application_Software_Date*	0...FFFF $FFFF_h$	-	unsigned 32	rw
	2	*Application_Software_Time*	0...0526 $5C00_h$	-	unsigned 32	rw

Description of the variable:

Application_Software_Date Date of the generation of the firmware used, specified in number of days since 1. January 1984

Application_Software_Time Time of the generation of the firmware used, specified in milliseconds since midnight.

9. References

[1] CiA 301 Standard
CANopen Application Layer and Communication Profile V4.2.0 (12.2011)

[2] CiA Draft Standard Proposal 302
CANopen Additional Application Layer Functions V4.1 (04.2010)

[3] CiA Draft Recommendation 303
CANopen Additional Specification V1.3 (08.2006)

[4] CiA Draft Standard Proposal 404
Device Profile for Measuring Devices and Closed-Loop Controllers V1.3.0 (03.2009)

[5] CiA Draft Recommendation 303 V1.4 (08.2006)
CANopen Additional specification, Part 2: Representation of SI units and prefix

[6] Phoenix Contact GmbH & Co. KG, Blomberg.
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector - FKCT-2,5/4-ST KMGY - 1921900, downloaded 2014-10-29

[7] Phoenix Contact GmbH & Co. KG, Blomberg.,
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector - FK-MCP 1,5/ ...-STF-3,81 - 1851261, downloaded 2014-10-29

[8] Phoenix Contact GmbH & Co. KG, Blomberg.,
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector - MC 1,5/12-STF-3,81 - 1827800, downloaded 2014-11-29

10. Glossary

FV . . . . Field Value (ADC-raw data)

PGA . . . . . Programmable Gain Amplifier (adjustable amplifier in the A/D-converters)

PV . . . . Process Value (corrected user data of the thermocouple input)

R/W . . . . . Read/Write (read and write access)

11. Declaration of Conformity

EU-KONFORMITÄTSERKLÄRUNG
EU DECLARATION OF CONFORMITY

Adresse / *Address*
esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

esd erklärt, dass das Produkt
esd declares, that the product
CAN-CBX-THERMO

Typ, Modell, Artikel-Nr.
Type, Model, Article No.
C.3034.02

die Anforderungen der Normen
fulfills the requirements of the standards
EN 61000-6-2:2005,
EN 61000-6-4:2007+A1:2011

gemäß folgendem Prüfbericht erfüllt.
according to test certificate.
H-K00-0435-11

Das Produkt entspricht damit der EU-Richtlinie „EMV"
Therefore the product corresponds to the EU Directive 'EMC'
2014/30/EU

Das Produkt entspricht der EU-Richtlinie „RoHS"
The product corresponds to the EU Directive 'RoHS'
2011/65/EU

Diese Erklärung verliert ihre Gültigkeit, wenn das Produkt nicht den Herstellerunterlagen entsprechend eingesetzt und betrieben wird, oder das Produkt abweichend modifiziert wird.
This declaration loses its validity if the product is not used or run according to the manufacturer's documentation or if non-compliant modifications are made.

Name / *Name* T. Ramm
Funktion / *Title* CE-Koordinator / *CE Coordinator*
Datum / *Date* Hannover, 2014-10-17

Rechtsgültige Unterschrift / *authorized signature*

I:\Texte\Doku\MANUALS\CAN\CBX\THERMO\EG-Konformitätserklärung\CAN-CBX-THERMO_EG-Konformitaetserklaerung_2014-10-17.odt

12. Order Information

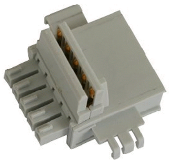
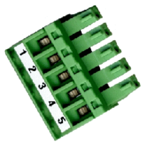
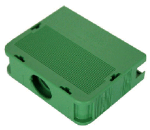
Type	Features	Order No.
CAN-CBX-THERMO	CAN-CBX-module with 4 thermocouple interfaces with cold-junction compensation	C.3034.02
Accessories		
CAN-CBX-TBUS 	Mounting-rail bus connector of the CBX-InRailBus for CAN-CBX-modules, (one bus connector is included in delivery of the CAN-CBX-module) Phoenix Contact order information: ME 22,5 TBUS 1,5/5-ST-3,81 KMGY (2713722)	C.3000.01
CAN-CBX-TBUS-Connector 	Terminal plug of the CBX-InRailBus for the connection of the +24 V power supply voltage and the CAN interface Female type Phoenix Contact order information: MCVR 1,5/ 5-STF-3,81 AU - (1846631)	C.3000.02
CAN-CBX-TBUS-Connection adapter 	Terminal plug of the CBX-InRailBus for the connection of the +24 V power supply voltage and the CAN interface Male type Phoenix Contact order information: IMC 1,5/ 5-ST-3,81 AU (1943276)	C.3000.03
CABLE HOUSING 	Cable housing for thermocouple inputs to improve the cold junction compensation Phoenix Contact order information: KGG MC1,5/12	C.3000.04

Table 13: Order information

PDF Manuals
For availability of English manuals see the following table.
Please download the manuals as PDF documents from our esd website www.esd.eu for free.

Manuals		Order No.
CAN-CBX-THERMO-ME	Manual in English	C.3034.21

Table 14: Available manuals

Printed Manuals
If you need a printout of the manual additionally, please contact our sales team: sales@esd.eu for a quotation. Printed manuals may be ordered for a fee.